The EMPIR initiative is co-funded by the European Union's Horizon 2020 research and innovation programme and the EMPIR Participating States

MetHyInfra

EMPIR CALL 2020

Metrology infrastructure for high-pressure gas and liquified hydrogen flows

CFD Workshop

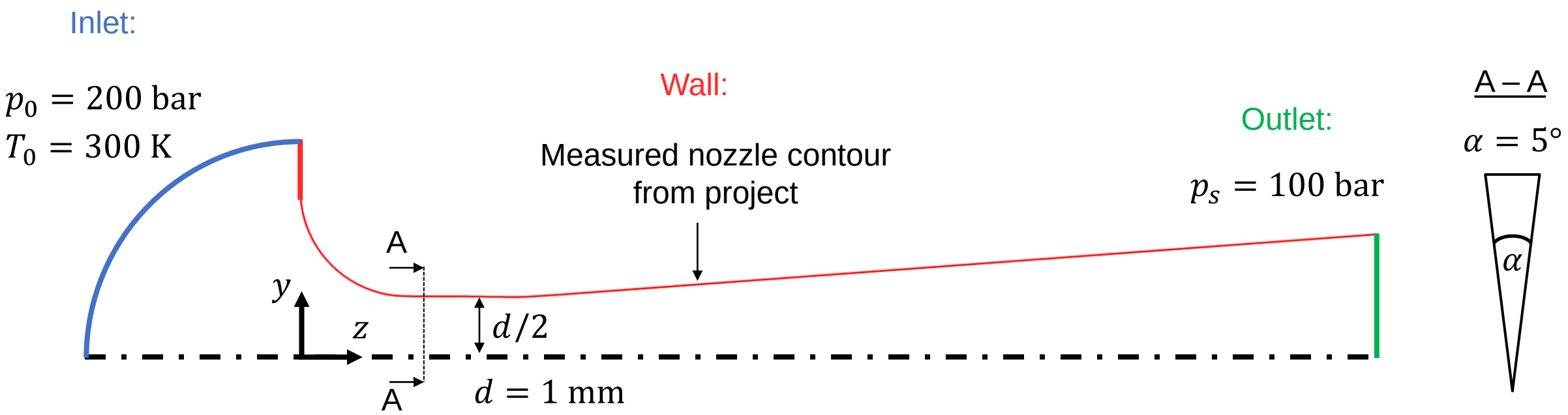
Part 3: Tutorial case of a critical nozzle

*"**FROM A**djusting the mesh **TO** visuali**Z**ing the flow field"*

Sebastian Weiss
Working Groups 8.41 “Modelling and Simulation” and 1.45 “Hydrogen Quantity Metering”
Physikalisch-Technische Bundesanstalt (PTB), Berlin, Germany

Tutorial case description

High-pressure hydrogen flow through a cylindrical non-ideal critical nozzle



- Simulation is conducted with OpenFOAM version:
  ESI OpenCFD Release OpenFOAM® v2012 (20 12)

Agenda (Part 1)

Pre-processing

1. Geometry creation
2. Mesh creation
3. Numerical setup
   a) Boundary conditions
   b) Turbulence model
   c) Real gas model
   d) Numerical settings

OpenFOAM folder structure

- 0
- constant
- system
- nozzle.foam

0 /

alphat	turbulent thermal diffusivity
k	turbulence kinetic energy
nut	turbulent viscosity
omega	turbulence specific dissipation rate
p	pressure
T	temperature
U	velocity

constant /

thermophysicalProperties	thermophysical properties
turbulenceProperties	turbulence properties

system /

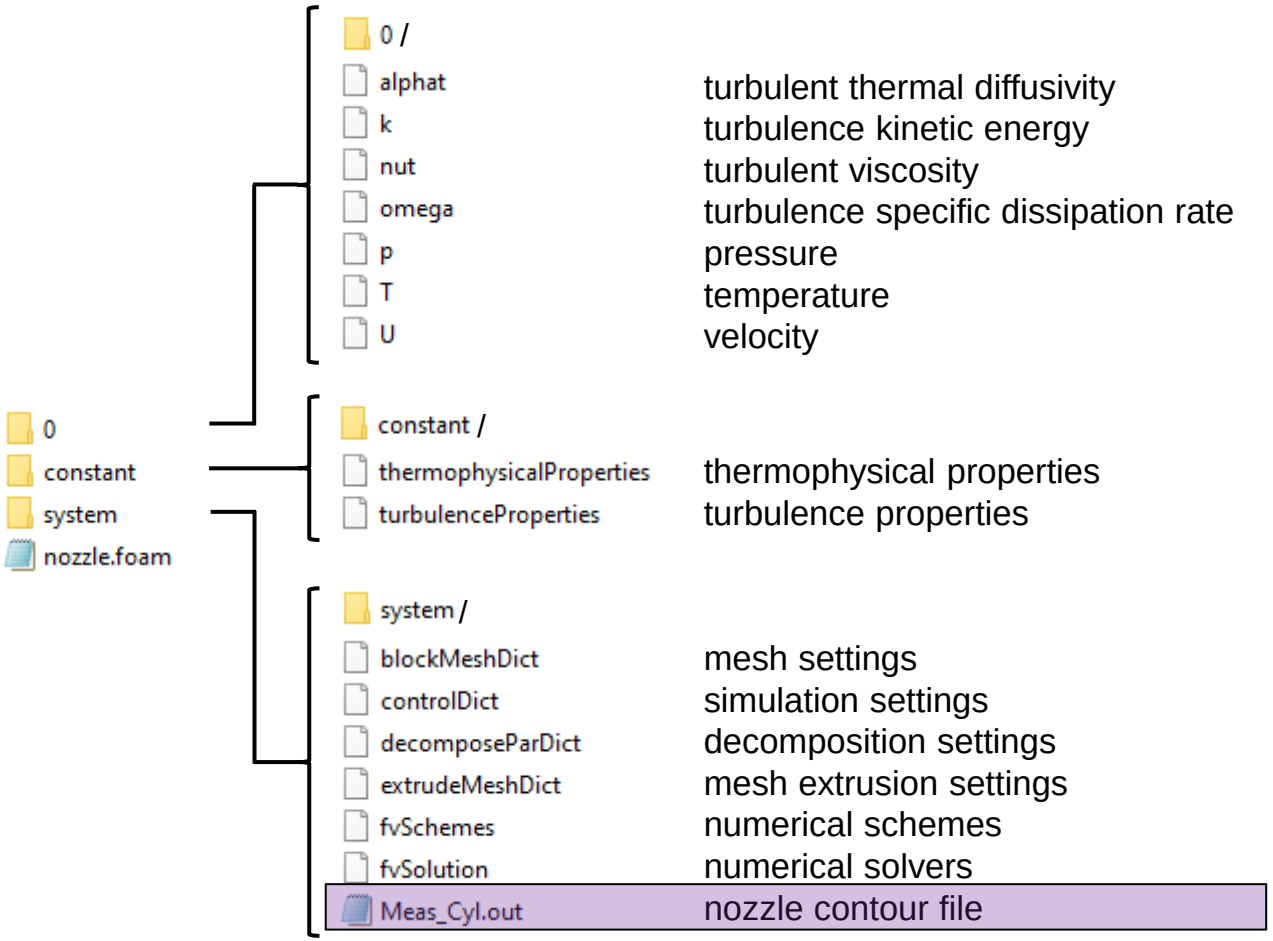
blockMeshDict	mesh settings
controlDict	simulation settings
decomposeParDict	decomposition settings
extrudeMeshDict	mesh extrusion settings
fvSchemes	numerical schemes
fvSolution	numerical solvers
Meas_Cyl.out	nozzle contour file

Agenda (Part 1)

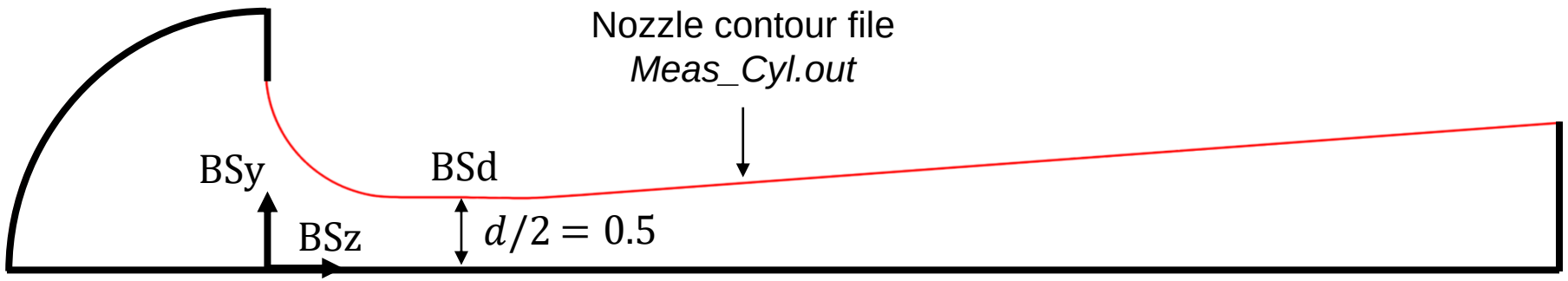
Pre-processing

1. Geometry creation
2. Mesh creation
3. Numerical setup
   a) Boundary conditions
   b) Turbulence model
   c) Real gas model
   d) Numerical settings

OpenFOAM folder structure



1. Geometry creation



- Nozzle contour file is created using Python script
- Specific file format required for later use in meshing tool
  - BSz: vector of axial nozzle coordinates
  - BSy: vector of radial nozzle coordinates
  - BSd: index of throat diameter position
- Note: throat diameter is always normalized to 1 in this file
- Functions are available for:
  1. Toroidal nozzles (*Tor_ideal*)
  2. Cylindrical nozzles (*Cyl_ideal*)
  3. Measured nozzles (*Meas_CFVN*)
  4. Creation of geometry file (*create_geometry_file*)

Nozzle contour file format
(for use in blockMesh)

$$\mathrm{BSz}\ (\ z_0\ z_1\ \ldots z_{nz-1}\ );$$

$$\mathrm{BSy}\ (\ y_0\ y_1\ \ldots y_{nz-1}\ );$$

$$\mathrm{BSd}\ \ \mathrm{index_d};$$

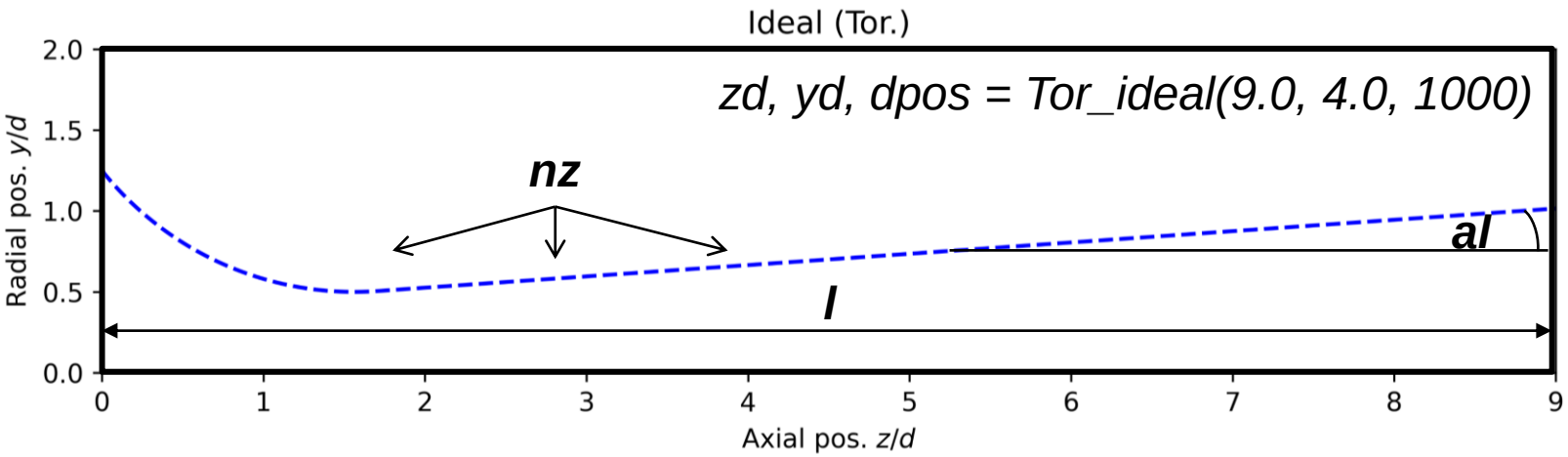
➢ **Let's have a look at how to create the geometry file *Meas_Cyl.out* for our tutorial case**

1. Geometry creation

Recap of the geometry functions

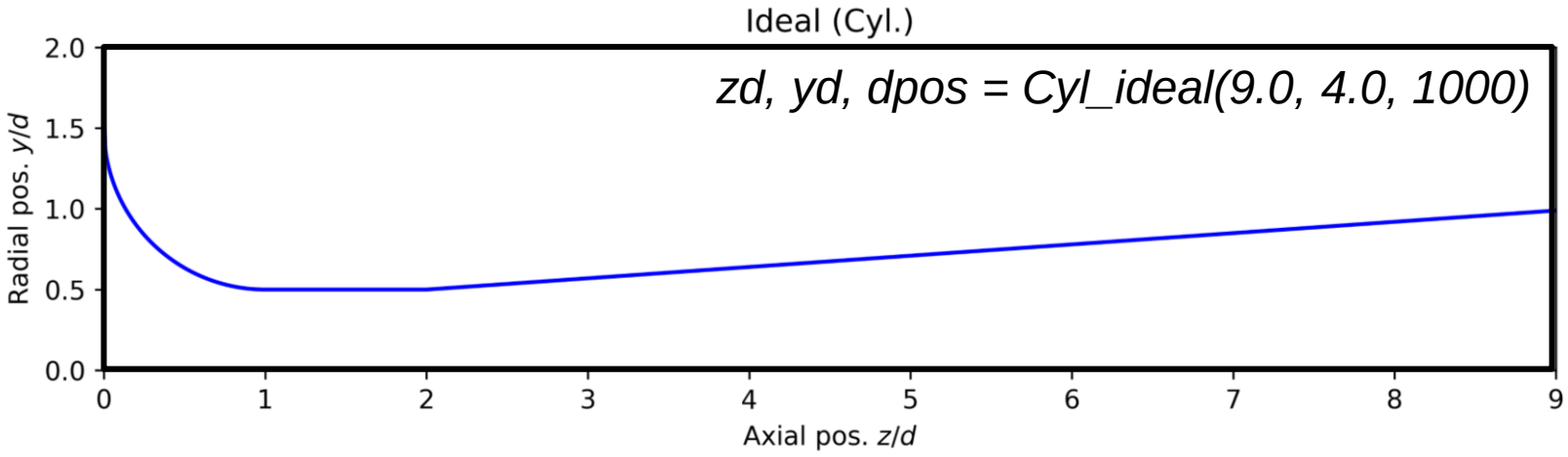
Ideal toroidal nozzle:
Function: ***Tor_ideal(l, al, nz)***
Parameters:

- ***l*** nozzle length (in d)
- ***al*** diffusor angle (in °)
- ***nz*** number of points



Ideal cylindrical nozzle:
Function: ***Cyl_ideal(l, al, nz)***
Parameters:

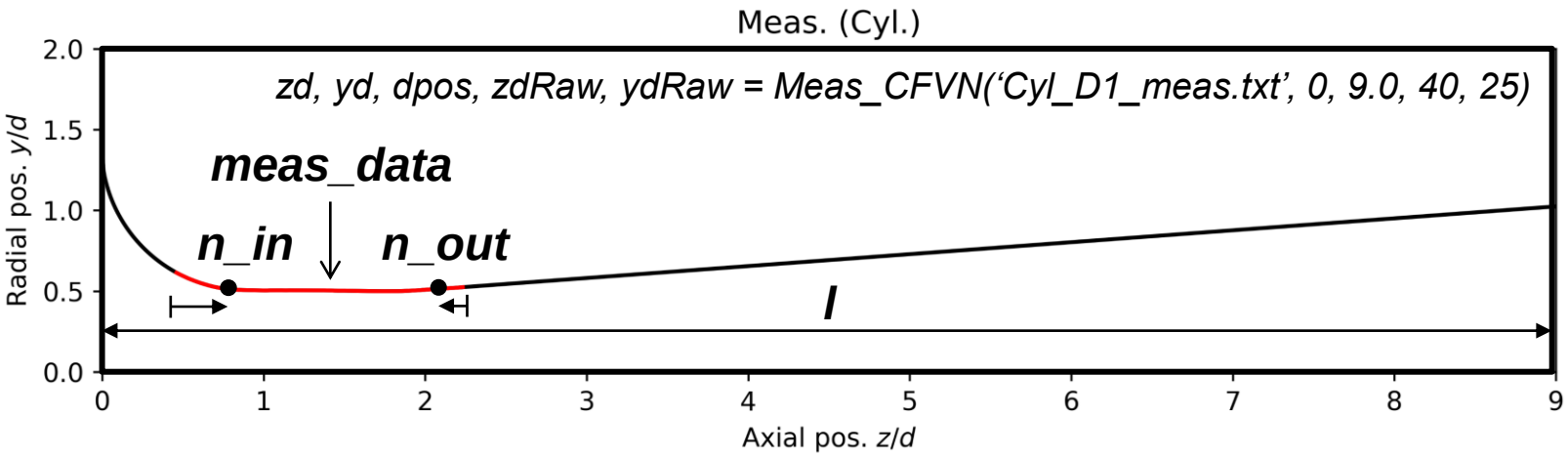
- ***l*** nozzle length (in d)
- ***al*** diffusor angle (in °)
- ***nz*** number of points



1. Geometry creation

Recap of the geometry functions

Measured nozzle:
Function:
Meas_CFVN(meas_data, NType, l, n_in, n_out)



Parameters:

- ***meas_data*** file containing measured contour data (list of z and y positions, tab-separated)
- ***NType*** nozzle type (0 if cylindrical, 1 if toroidal)
- ***l*** nozzle length (in d)
- ***n_in*** point index for inlet circle (in *meas_data*)
- ***n_out*** point index for outlet slope (counted from last index in *meas_data*)

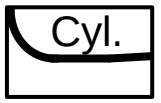
meas_data

From CMM measurement:

Cyl_D1_meas.txt

z	r
-0.9518087	0.614441116
-0.9429014	0.609777601
-0.9339942	0.605116062
-0.9250869	0.600719494
-0.9161797	0.596532892

NType

0 

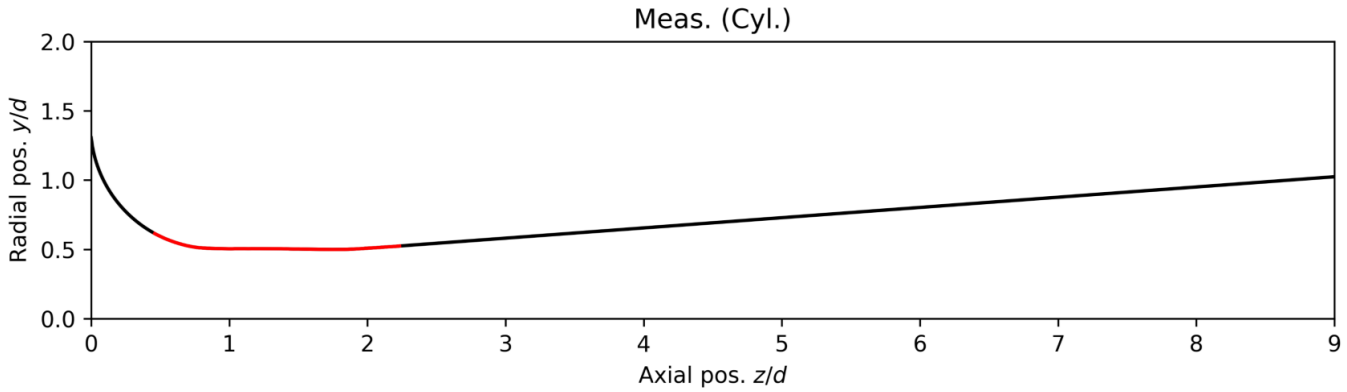
1 

1. Geometry creation

Recap of the geometry functions

Creation of geometry file:
Function: ***create_geometry_file(name, BSz, Bsy, BSd)***
Parameters:

- ***name*** name of output file (here: *Meas_Cyl.out*)
- ***BSz*** vector of axial nozzle coordinates (already created)
- ***BSy*** vector of radial nozzle coordinates (already created)
- ***BSd*** index of throat diameter position (already created)



➢ **Geometry is created ✓**

Output file (*Meas_Cyl.out*)

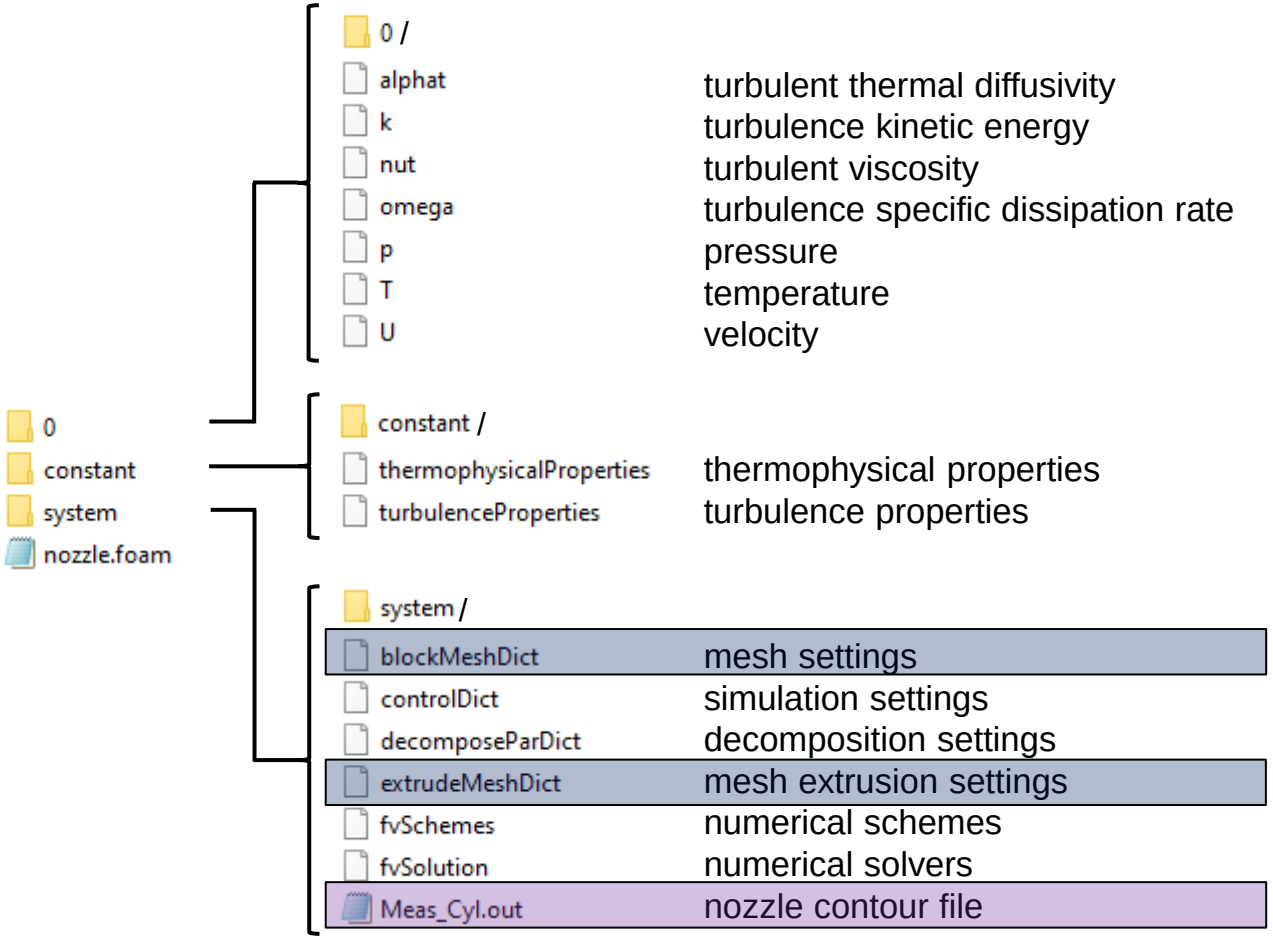
```
BSz ( 0.000000 0.008842 0.017        31857 0.940764 0.949671 0.958
.003115 1.012022 1.020930 1.6        .947282 1.956189 1.965096 1.97
2.018540 2.027446 2.036354 2.        2.957738 2.966585 2.975432 2.9
5 3.028512 3.037358 3.046205         3 3.966256 3.975103 3.983950 3
183 4.037030 4.045876 4.05472   ...  928 4.974774 4.983621 4.99246
36701 5.045548 5.054394 5.063        74446 5.983293 5.992139 6.000
.045219 6.054066 6.062912 6.6        .982964 6.991811 7.000657 7.0
7.053737 7.062584 7.071430 7.        7.991482 8.000329 8.009175 8.6
9 8.062255 8.071102 8.079948         3 9.000000  );

BSy ( 1.304710 1.241684 1.198        06056 0.505914 0.505739 0.505
.504587 0.504455 0.504787 0.5        .505329 0.505887 0.506235 0.5
0.510218 0.510659 0.511116 0.        3.578373 0.579027 0.579680 0.5
7 0.583600 0.584254 0.584907         2 0.652856 0.653509 0.654162 6
429 0.658082 0.658736 0.65938   ...  584 0.727338 0.727991 0.72864
31911 0.732565 0.733218 0.733        01167 0.801820 0.802473 0.803
.806393 0.807047 0.807700 0.8        .875649 0.876302 0.876956 0.87
0.880876 0.881529 0.882182 0.        3.950131 0.950784 0.951438 0.9
5 0.955358 0.956011 0.956665         3 1.024613  );

BSd 201;
```

Agenda (Part 1)

Pre-processing

1. Geometry creation
2. Mesh creation
3. Numerical setup
   a) Boundary conditions
   b) Turbulence model
   c) Real gas model
   d) Numerical settings

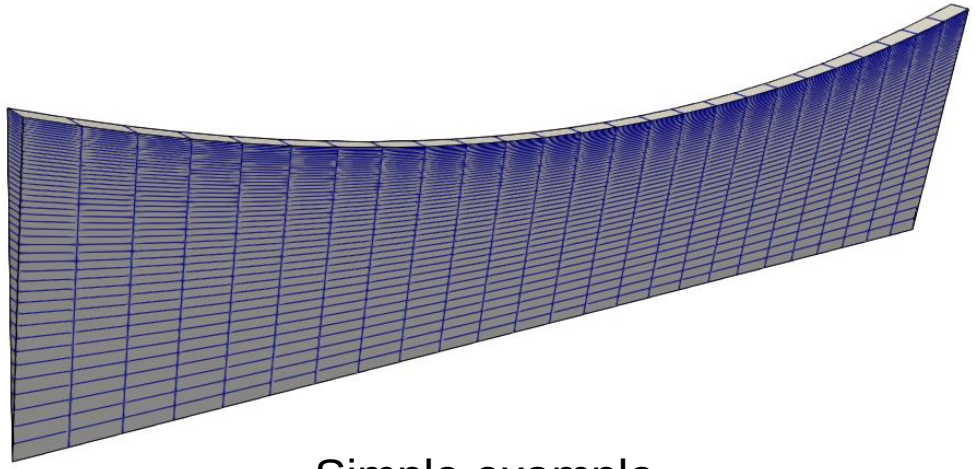
OpenFOAM folder structure



2. Mesh creation

- Mesh is created using **blockMesh** utility in OpenFOAM
- **blockMesh** generates hexaedral meshes from geometry using block structure
- Mesh information is written in *blockMeshDict* file
- We will create the mesh for the measured cylindrical nozzle geometry with a blockMesh-based nozzle meshing tool

➢ But first, let's have a look at a simple example to understand the blockMesh structure



Simple example

2. Mesh creation

Simple example (*blockMeshDict* file)

scale: Scaling factor (0.001 scales to mm)

```
scale 0.001;
```

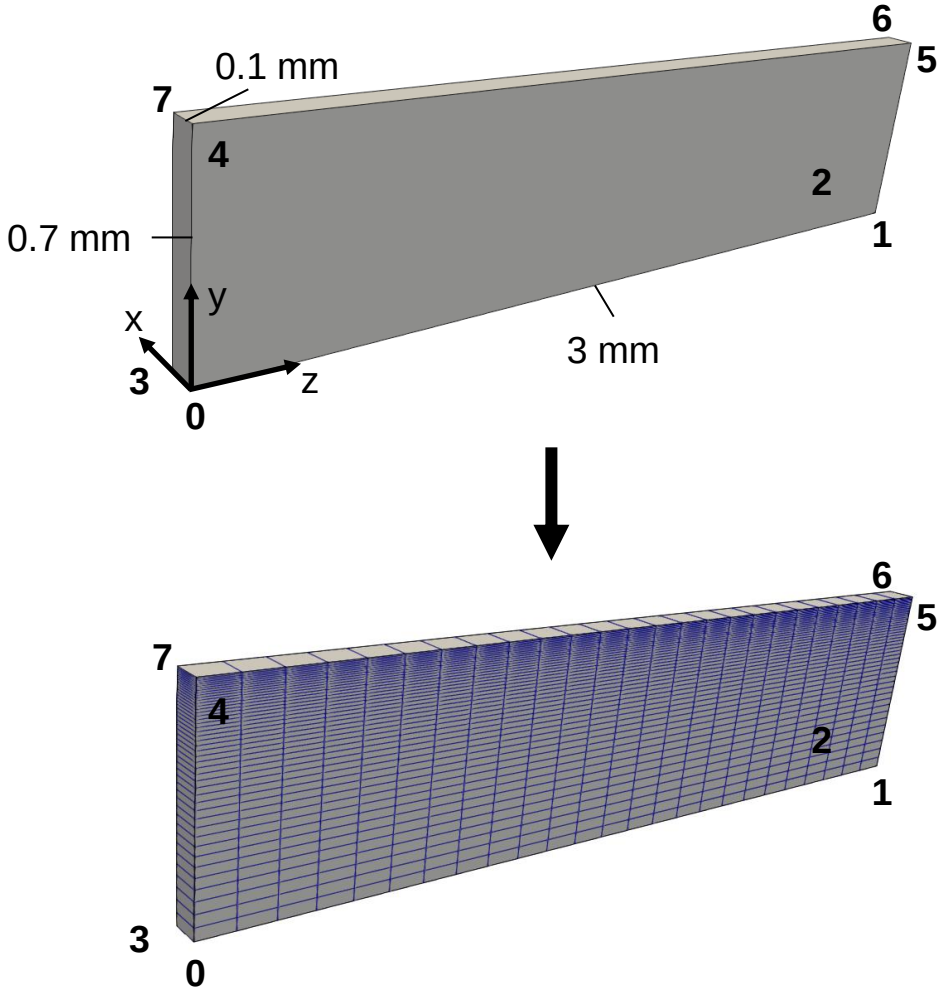
vertices: List of vertex coordinates

```
vertices
(
        (0.0 0.0 0.0) //0
        (0.0 0.0 3.0) //1
        (0.1 0.0 3.0) //2
        (0.1 0.0 0.0) //3
        (0.0 0.7 0.0) //4
        (0.0 0.7 3.0) //5
        (0.1 0.7 3.0) //6
        (0.1 0.7 0.0) //7
);
```

blocks: Ordered list of vertex labels and mesh size

```
blocks
(
        hex (0 1 2 3 4 5 6 7) (25 1 45) simpleGrading (1 1 0.1)
);
```

Vertex numbers | Number of cells in each direction | Cell expansion ratios



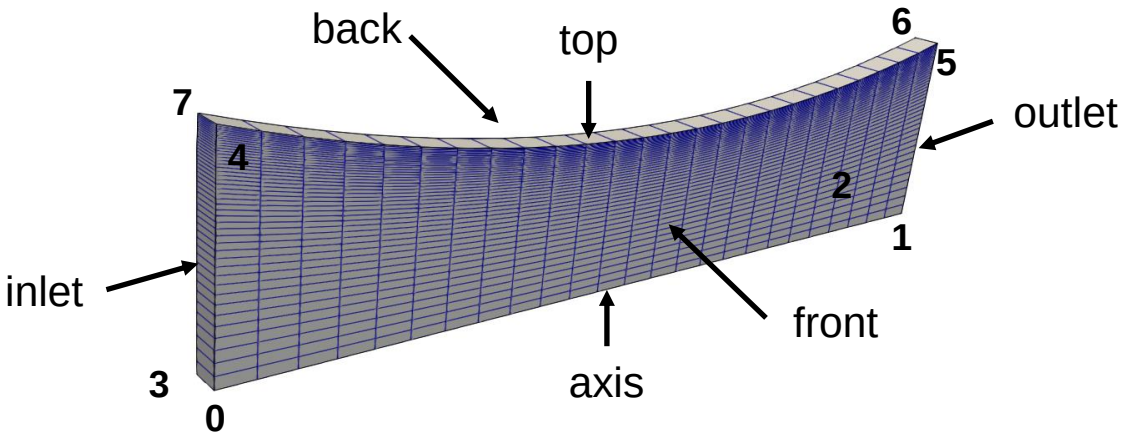
2. Mesh creation

edges: Used to define arc or spline edges

```
edges
(
        arc 4 5 (0.0 0.5 1.5)
        arc 7 6 (0.1 0.5 1.5)
);
```

boundary: List of boundary faces and type

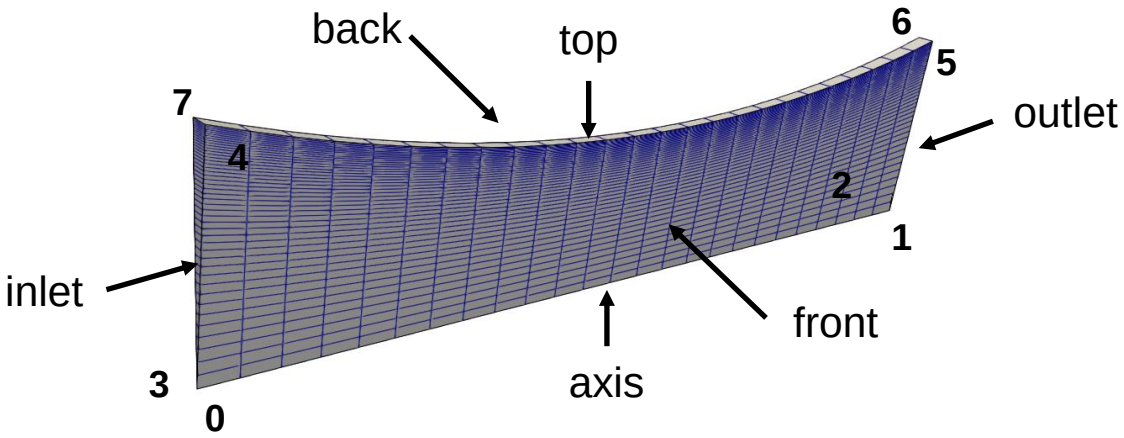
```
boundary
(

);
```



2. Mesh creation

extrudeMeshDict: Used to create a 5° wedge sector mesh

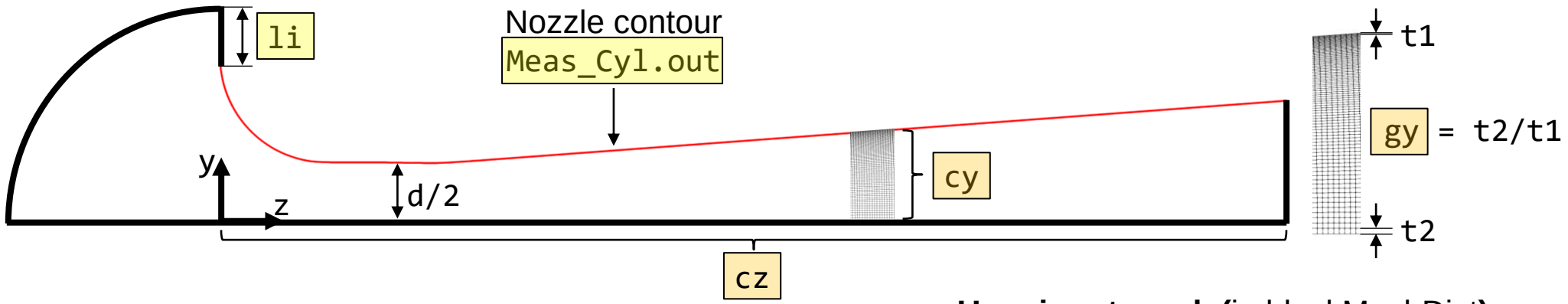
```
constructFrom   patch;
sourceCase      "<case>";

sourcePatches   (front);
exposedPatchName back;

extrudeModel    wedge;

sectorCoeffs
{
    axisPt          (0 0 0);
    axis            (0 0 1);
    angle           5;
}

flipNormals     false;

mergeFaces      false;
```



➢ Now we create the mesh for our tutorial case using more advanced nozzle meshing tool

2. Mesh creation

2D meshing tool to create an arbitrary nozzle contour



Input variables

Geometry

- *sc* scaling factor (sc = 0.001 → d = 1 mm)
- *li* inlet length (in d)
- *Nozzle contour file*

Mesh

- *cy* number of cells in the y-direction
- *cz* number of cells in the z-direction
- *gy* grading in the y-direction

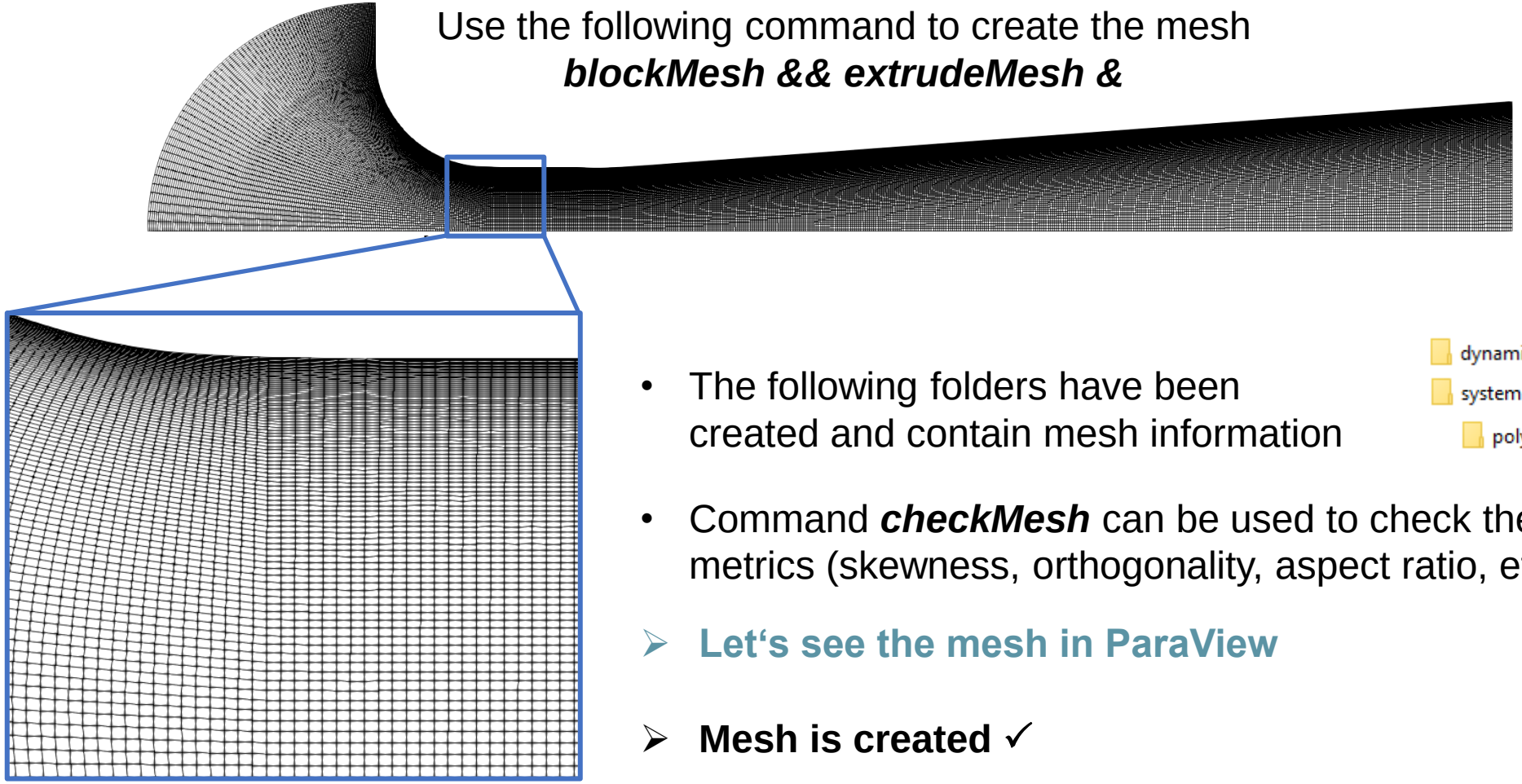
User input mask (in blockMeshDict)

```
// * * * * * * * * * * * MESH GENERATION TOOL OF 2D NOZZLE * * * * * * * * *   //
//                                                                             //
// * * * * * * * * * * * * * * * INPUT PARAMETERS  * * * * * * * * * * * *   //
//                                                                             //
// - - - - - - - - - - - - - - - - - - Geometry - - - - - -  - - - - - - - - -   //
//                                                                             //
sc 0.001;      // Scaling factor (sc = 0.001 ==> d = 1 mm)                     //
li 0.5;        // Inlet length (in d)                                          //
// list of z and y coordinates of nozzle contour                               //
#include "Meas_Cyl.out";                                                       //
//                                                                             //
// - - - - - - - - - - - - - - - - - - - - Mesh - - - - - - - - - - - - - - -   //
//                                                                             //
cy 90;         // Number of cells in y of nozzle                               //
cz 600;        // Number of cells in z of nozzle                               //
gy 10;         // Grading in y towards nozzle wall                             //
//                                                                             //
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *   //
```

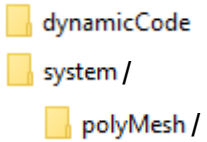
2. Mesh creation

2D meshing tool to create an arbitrary nozzle contour

Use the following command to create the mesh
blockMesh && extrudeMesh &



- The following folders have been created and contain mesh information



- Command ***checkMesh*** can be used to check the mesh metrics (skewness, orthogonality, aspect ratio, etc.)

➢ **Let‘s see the mesh in ParaView**

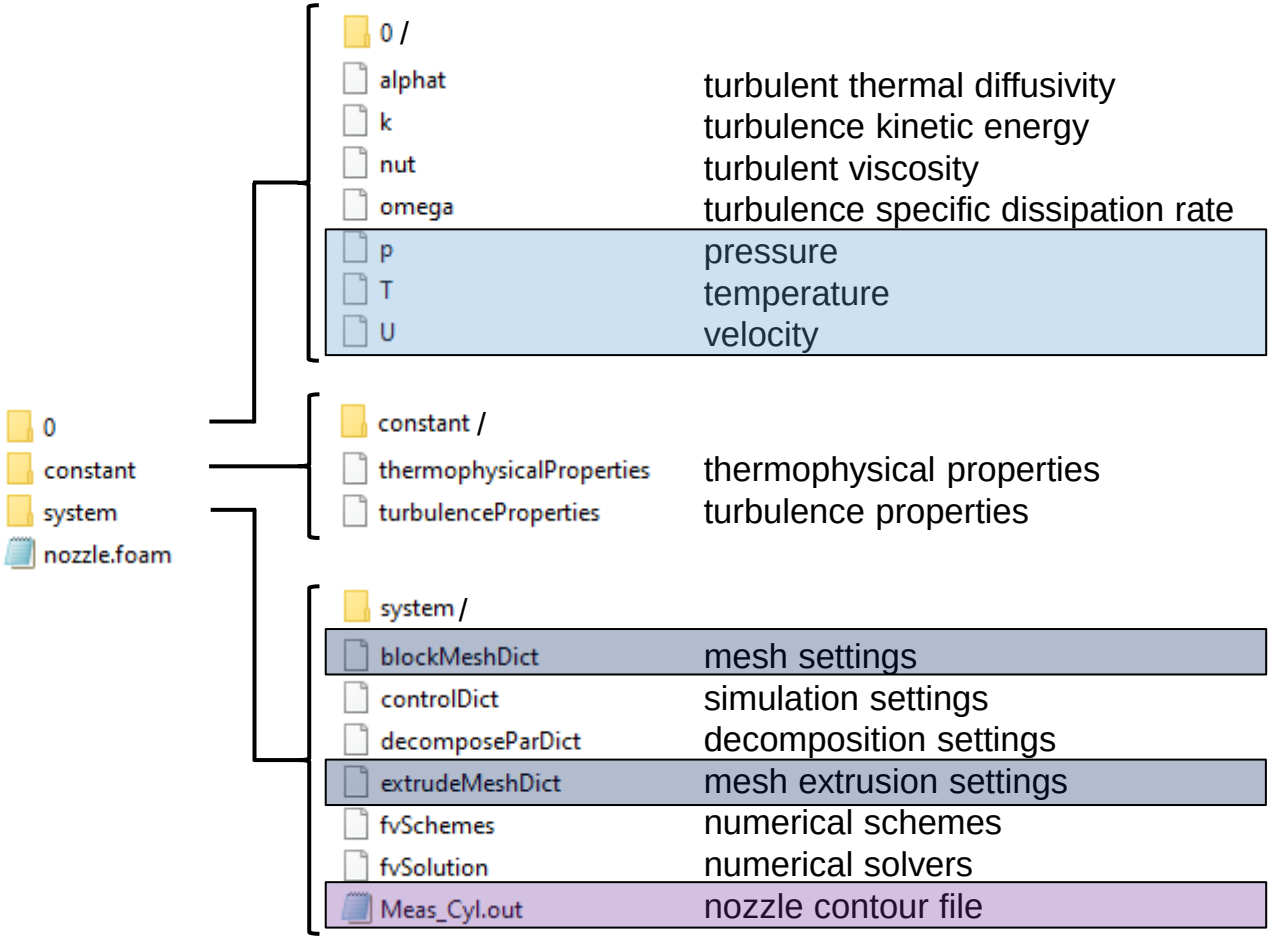
➢ **Mesh is created ✓**

Agenda (Part 1)

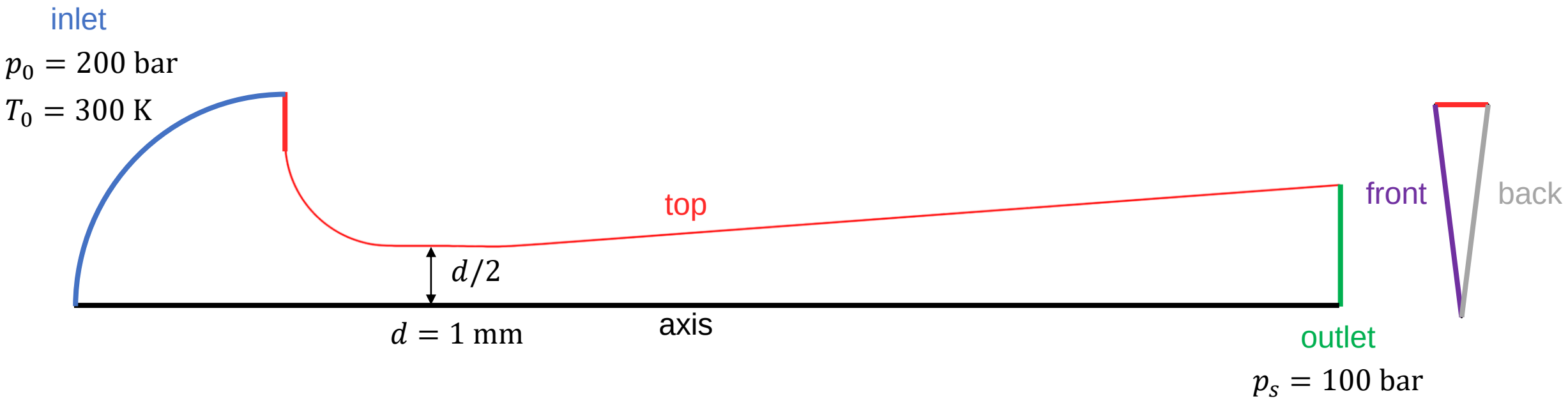
Pre-processing

1. Geometry creation
2. Mesh creation
3. Numerical setup
   a) Boundary conditions
   b) Turbulence model
   c) Real gas model
   d) Numerical settings

OpenFOAM folder structure



3.a) Boundary conditions



- **First, let‘s estimate the throat Reynolds number**

3.a) Boundary conditions

Estimation of throat Reynolds number

- Use definitions from ISO 9300
- For reasons of simplicity, assume ideal gas and isentropic, 1D flow

Throat Reynolds number

2.4.2
throat Reynolds number
Re_{nt}
dimensionless parameter calculated from the gas flow-rate and the gas dynamic viscosity at nozzle inlet stagnation conditions

NOTE The characteristic dimension is taken as the throat diameter at stagnation conditions. The throat Reynolds number is given by the formula:

$$Re_{\mathrm{nt}} = \frac{4 q_m}{\pi d \mu_0}$$

$$\rightarrow Re_{nt,th,i} = \frac{4\dot{m}_{th,i}}{\pi d \mu_0}$$

Ideal mass flow rate

4.2 Flow-rate under ideal conditions

$$q_{mi} = \frac{A_{\mathrm{nt}} C_{*i} p_0}{\sqrt{\left(\frac{R}{M}\right) T_0}}$$

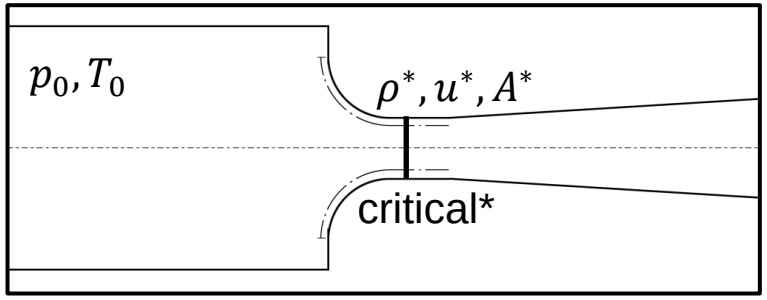
where

$$C_{*i} = \sqrt{\gamma \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}}$$

$$\rightarrow \dot{m}_{th,i} = C_i^* \cdot \frac{\pi}{4} d^2 \cdot \frac{p_0}{\sqrt{\left(\frac{R}{M}\right) T_0}}$$

- We often take this expression for granted
- ➢ But how can we actually derive this formulation?

3.a) Boundary conditions



Mass flow rate

$$\dot{m}_{th,i} = C_i^* \cdot \frac{p_0}{\sqrt{R_M T_0}} \cdot \frac{\pi}{4} d^2$$

How can we derive this equation?

1. Start with simple definition:

$$\dot{m}_{th,i} = \rho^* u^* A^*$$

2. Assume:
   - Ideal gas
   - Reversible process
   - Adiabatic process
   - 1D flow

Critical area

$$A^* = \frac{\pi}{4} d^2 \quad ✓$$

Critical velocity

Critical Mach number

$$M^* = \frac{u^*}{a^*} = 1$$

$$u^* = a^* = \sqrt{\kappa R_M T^*}$$

Critical temperature is still unknown

Isenthalpic process (constant total enthalpy)

$$h_0 = h^* + \frac{1}{2} u^{*2}$$

$$c_p T_0 = c_p T^* + \frac{1}{2} \kappa R_M T^*$$

$$T_0 = T^* \left(1 + \frac{1}{2} \cdot \frac{\kappa R_M}{c_p}\right)$$

$$T^* = T_0 \left(\frac{2}{\kappa + 1}\right)$$

$$u^* = \sqrt{\frac{2\kappa}{\kappa + 1} R_M T_0}$$

Ideal gas

$$\kappa = \frac{c_p}{c_v} \quad (\gamma \text{ in ISO 9300})$$

$$R_M = c_p - c_v$$

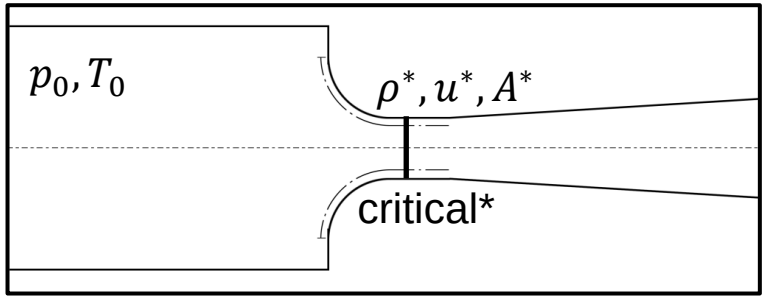
Rearranging

$$1 + \frac{1}{2} \cdot \frac{\kappa R_M}{c_p}$$

$$= 1 + \frac{1}{2} \cdot \frac{c_p (c_p - c_v)}{c_v \cdot c_p}$$

$$= 1 + \frac{\kappa - 1}{2} = \frac{\kappa + 1}{2}$$

3.a) Boundary conditions



Mass flow rate

$$\dot{m}_{th,i} = C_i^* \cdot \frac{p_0}{\sqrt{R_M T_0}} \cdot \frac{\pi}{4} d^2$$

How can we derive this equation?

1. Start with simple definition:

$$\dot{m}_{th,i} = \rho^* u^* A^*$$

2. Assume:
   - Ideal gas
   - Reversible process
   - Adiabatic process
   - 1D flow

Critical area

$$A^* = \frac{\pi}{4} d^2$$

Critical velocity

$$u^* = \sqrt{\frac{2\kappa}{\kappa + 1} R_M T_0}$$

Critical density

$$\rho^* = \frac{p_0}{R_M T_0} \left(\frac{2}{\kappa + 1} \right)^{\frac{1}{\kappa - 1}}$$

Isentropic process
(constant entropy)

$$\frac{p_0}{\rho_0^\kappa} = \frac{p^*}{\rho^{*\kappa}}$$

$$\frac{R_M T_0}{\rho_0^{\kappa-1}} = \frac{R_M T^*}{\rho^{*\kappa-1}}$$

Ideal
gas law

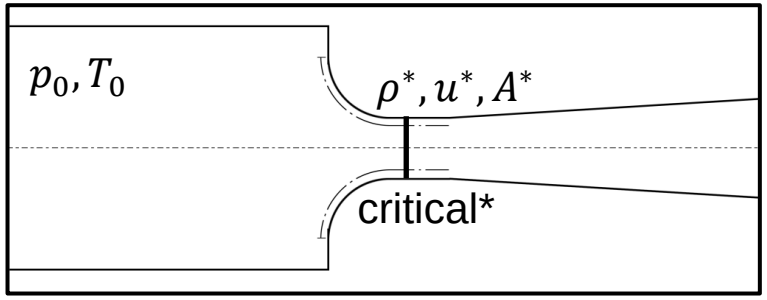
$$p = \rho R_M T$$

$$\left(\frac{\rho^*}{\rho_0} \right)^{\kappa-1} = \left(\frac{T^*}{T_0} \right) \longrightarrow \rho^* = \rho_0 \left(\frac{T^*}{T_0} \right)^{\frac{1}{\kappa-1}}$$

With critical
temperature

$$T^* = T_0 \left(\frac{2}{\kappa + 1} \right)$$

3.a) Boundary conditions



Mass flow rate

$$\dot{m}_{th,i} = C_i^* \cdot \frac{p_0}{\sqrt{R_M T_0}} \cdot \frac{\pi}{4} d^2$$

How can we derive this equation?

1. Start with simple definition:

$$\dot{m}_{th,i} = \rho^* u^* A^*$$

2. Assume:
   - Ideal gas
   - Reversible process
   - Adiabatic process
   - 1D flow

Critical area

$$A^* = \frac{\pi}{4} d^2$$

Critical velocity

$$u^* = \sqrt{\frac{2\kappa}{\kappa+1} R_M T_0}$$

Critical density

$$\rho^* = \frac{p_0}{R_M T_0} \left(\frac{2}{\kappa+1}\right)^{\frac{1}{\kappa-1}}$$

$$\dot{m}_{th,i} = \frac{p_0}{R_M T_0} \left(\frac{2}{\kappa+1}\right)^{\frac{1}{\kappa-1}} \cdot \sqrt{\frac{2\kappa}{\kappa+1} R_M T_0} \cdot \frac{\pi}{4} d^2$$

Rearranging

$$\dot{m}_{th,i} = \underbrace{\sqrt{\kappa \cdot \left(\frac{2}{\kappa+1}\right)^{\frac{\kappa+1}{\kappa-1}}}}_{C_i^*} \cdot \frac{p_0}{\sqrt{R_M T_0}} \cdot \frac{\pi}{4} d^2$$

ISO 9300

$$C_{*i} = \sqrt{\gamma \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}}$$

3.a) Boundary conditions

MetHyInfra

Estimation of throat Reynolds number

- Throat Reynolds number

$$Re_{nt,th,i} = \frac{4\dot{m}_{th,i}}{\pi d \mu_0}$$

- Ideal mass flow rate

$$\dot{m}_{th,i} = C_i^* \cdot \frac{\pi}{4} d^2 \cdot \frac{p_0}{\sqrt{\left(\frac{R}{M}\right) T_0}}$$

- Ideal critical flow factor $(\kappa = 1.4)$

$$C_i^* = \sqrt{\kappa \cdot \left(\frac{2}{\kappa + 1}\right)^{\frac{\kappa+1}{\kappa-1}}} \approx 0.68473$$

$$\boxed{\rightarrow Re_{nt,th,i} = \frac{C_i^* \cdot d \cdot p_0}{\mu_0 \sqrt{\frac{RT_0}{M}}} \approx 1.326 \cdot 10^6}$$

➢ Let‘s set the boundary conditions in OpenFOAM

Dynamic viscosity (REFPROP):

$$\mu_0(p_0, T_0) = 9.2823 \cdot 10^{-6}\ \mathrm{Pa \cdot s}$$

Universal gas constant:

$$R = 8.3144626 \frac{\mathrm{J}}{\mathrm{molK}}$$

Molar mass:

$$M = 0.002016 \frac{\mathrm{kg}}{\mathrm{mol}}$$

3.a) Boundary conditions

Pressure *p*

Mass [kg] Time [s] Quantity [mol] Luminous intensity [cd]

```
dimensions      [1 -1 -2 0 0 0 0];
```

= Pascal [$kgm^{-1}s^{-2}$]

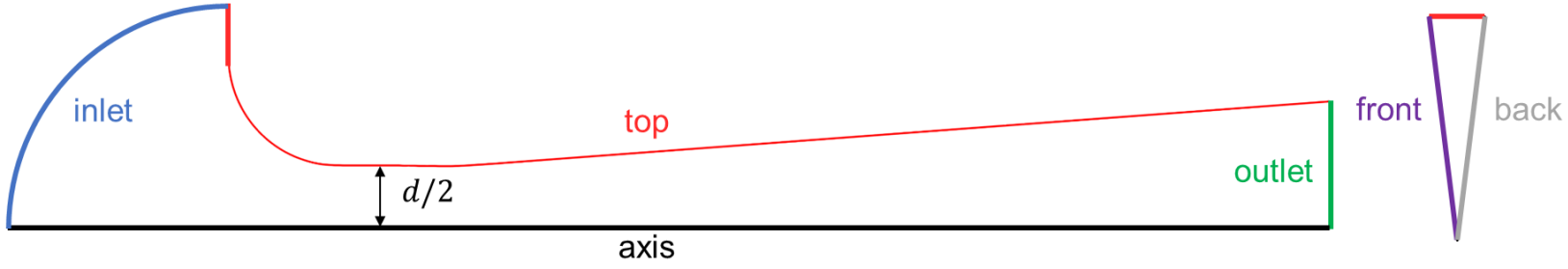
Length [m] Temperature [K] Current [A]

```
internalField   uniform 200e+05;
```

= Initial distribution inside of the nozzle (200 bar)

```
boundaryField
```

= Actual boundary conditions for each patch face



3.a) Boundary conditions

Pressure *p*

```
dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 200e+05;

boundaryField
{
    inlet
    {
        type            totalPressure;
        p0              uniform 200e+05;
        value           uniform 200e+05;
    }
    outlet
    {
        type            fixedMean;
        meanValue       100e+05;
        value           uniform 100e+05;
    }
    top
    {
        type            zeroGradient;
    }
    axis
    {
        type            empty;
    }
    front
    {
        type            wedge;
    }
    back
    {
        type            wedge;
    }
}
```

Temperature *T*

```
dimensions      [0 0 0 1 0 0 0];

internalField   uniform 300.0;

boundaryField
{
    inlet
    {
        type            totalTemperature;
        T0              uniform 300.0;
        value           $internalField;
        gamma           1.4;
    }
    outlet
    {
        type            zeroGradient;
    }
    top
    {
        type            zeroGradient; //adiabatic
    }
    axis
    {
        type            empty;
    }
    front
    {
        type            wedge;
    }
    back
    {
        type            wedge;
    }
}
```

Velocity *U*

```
dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type            pressureInletOutletVelocity;
        value           $internalField;
    }
    outlet
    {
        type            zeroGradient;
    }
    top
    {
        type            fixedValue;     // noSlip;
        value           uniform (0 0 0);
    }
    axis
    {
        type            empty;
    }
    front
    {
        type            wedge;
    }
    back
    {
        type            wedge;
    }
}
```

Inlet:
- Total pressure
- Total temperature

Outlet:
- Static pressure

Wall:
- No-slip
- Adiabatic

Axis:
- Rotation axis

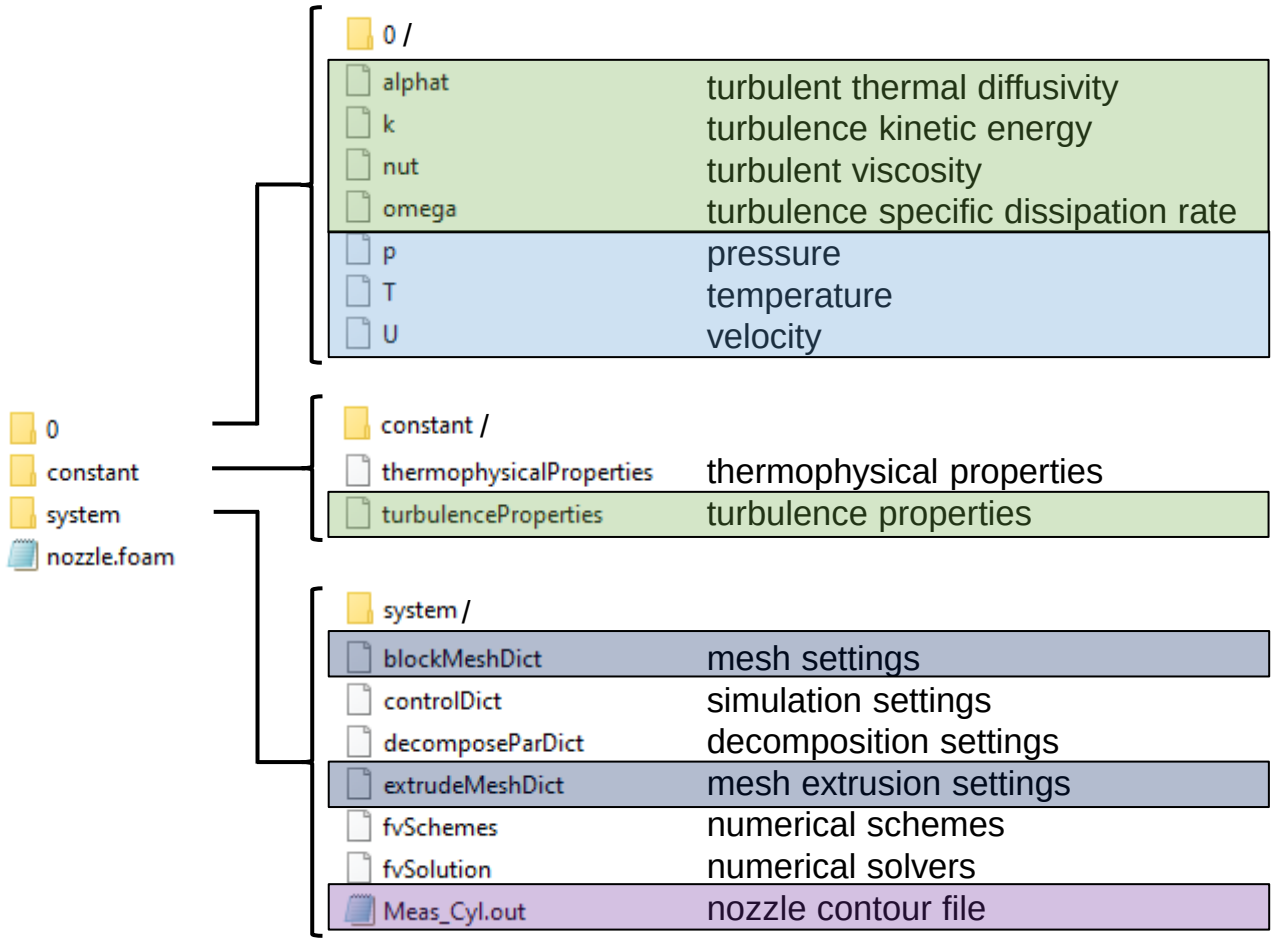
Front and Back:
- Symmetry

Agenda (Part 1)

Pre-processing

1. Geometry creation
2. Mesh creation
3. Numerical setup
   a) Boundary conditions
   b) Turbulence model
   c) Real gas model
   d) Numerical settings

OpenFOAM folder structure



3.b) Turbulence model

- The turbulence model is selected in the ***turbulenceProperties*** file

```
simulationType  banana;

RAS
{
    RASModel        kOmegaSST;

    turbulence      on;

    printCoeffs     on;
}
```

Banana?

‚Banana trick' output

```
Creating turbulence model

Selecting turbulence model type banana


--> FOAM FATAL IO ERROR: (openfoam-2012)
Unknown simulationType type banana

Valid simulationType types :
3(LES RAS laminar)
```

- The ‚banana trick' provokes an error message to show all valid types
- ➢ We select the *RAS* type

3.b) Turbulence model

- The turbulence model is selected in the ***turbulenceProperties*** file

```
simulationType  RAS;

RAS
{
    RASModel        kOmegaSST;

    turbulence      on;

    printCoeffs     on;
}
```

Reynolds-averaged simulation (RAS)

k-ω Shear Stress Transport (SST) model

- Two-equation model for turbulent kinetic energy *k* and turbulent specific dissipation rate *ω*
- Eddy viscosity model based on Boussinesq's hypothesis
- Combines strenghts of k-ω model (inner region of boundary layer) and k-ε model (free stream)

- We solve for four additional quantities:
  1. Turbulent kinetic energy *k*
  2. Turbulent specific dissipation rate *ω*
  3. Turbulent viscosity ν_t
  4. Turbulent thermal diffusivity α_t

Solved in additional transport equations

Modeled and included in momentum equation

Modeled and included in energy equation

3.b) Turbulence model

Turb. kin. energy *k*

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0.01215;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.01215;
    }

    outlet
    {
        type            zeroGradient;
    }


    top
    {
        type            kLowReWallFunction;
        value           $internalField;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

- How can we calculate the inlet value?
  - For isentropic turbulence, k_{in} can be estimated by:

$$k_{in} = \frac{3}{2}(Tu \cdot U_\infty)^2 \boxed{\approx 0.01215\ \frac{\mathrm{m}^2}{\mathrm{s}^2}}$$

  - Estimated turbulence intensity:

$$Tu = 0.5\ \%$$

  - Free stream velocity (based on preliminary calculations):

$$U_\infty \approx 18\ \frac{\mathrm{m}}{\mathrm{s}}$$

- Chosen wall function for *k*
  - Provides a wall constraint depending on the y^+ value

3.b) Turbulence model

Turb. kin. energy *k*

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0.01215;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.01215;
    }

    outlet
    {
        type            zeroGradient;
    }


    top
    {
        type            kLowReWallFunction;
        value           $internalField;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

Turb. diss. rate *omega*

```
dimensions      [0 0 -1 0 0 0 0];

internalField   uniform 1800.0;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1800.0;
    }

    outlet
    {
        type            zeroGradient;
    }


    top
    {
        type            omegaWallFunction;
        value           $internalField;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

- How can we calculate the inlet value?
  - ω_{in} is calculated as:

$$\omega_{in} = \frac{\sqrt{k_{in}}}{C_\mu^{0.25} \cdot l} \quad \boxed{\approx 1800\ \frac{1}{\mathrm{s}}}$$

  - Turbulence length scale (fully developed pipe flow):

$$l = 0.038 \cdot d_{in}$$

  - Inlet diameter:

$$d_{in} = 3 \cdot d$$

  - Model parameter:

$$C_\mu = 0.09$$

- Chosen wall function for ω
  - Provides a wall constraint depending on the y^+ value

3.b) Turbulence model

Turb. kin. energy *k*

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0.01215;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.01215;
    }

    outlet
    {
        type            zeroGradient;
    }


    top
    {
        type            kLowReWallFunction;
        value           $internalField;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

Turb. diss. rate *omega*

```
dimensions      [0 0 -1 0 0 0 0];

internalField   uniform 1800.0;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1800.0;
    }

    outlet
    {
        type            zeroGradient;
    }


    top
    {
        type            omegaWallFunction;
        value           $internalField;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

Turb. viscosity *nut*

```
dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            calculated;
        value           uniform 0;
    }

    outlet
    {
        type            calculated;
        value           uniform 0;
    }


    top
    {
        type            nutkWallFunction;
        value           uniform 0;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

- Chosen wall function for ν_t
  - Provides a wall constraint depending on the y^+ value

3.b) Turbulence model

Turb. kin. energy *k*

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0.01215;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.01215;
    }

    outlet
    {
        type            zeroGradient;
    }


    top
    {
        type            kLowReWallFunction;
        value           $internalField;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

Turb. diss. rate *omega*

```
dimensions      [0 0 -1 0 0 0 0];

internalField   uniform 1800.0;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1800.0;
    }

    outlet
    {
        type            zeroGradient;
    }


    top
    {
        type            omegaWallFunction;
        value           $internalField;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

- Chosen wall function for α_t
  - Provides a wall constraint depending on the y^+ value

Turb. therm. diff. *alphat*

```
dimensions      [1 -1 -1 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            calculated;
        value           uniform 0;
    }

    outlet
    {
        type            calculated;
        value           uniform 0;
    }

    top
    {
        type            compressible::alphatWallFunction;
        value           $internalField;
    }

    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

3.b) Turbulence model

Turb. kin. energy *k*

```
dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0.01215;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.01215;
    }

    outlet
    {
        type            zeroGradient;
    }


    top
    {
        type            kLowReWallFunction;
        value           $internalField;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

Turb. diss. rate *omega*

```
dimensions      [0 0 -1 0 0 0 0];

internalField   uniform 1800.0;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1800.0;
    }

    outlet
    {
        type            zeroGradient;
    }


    top
    {
        type            omegaWallFunction;
        value           $internalField;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

Turb. viscosity *nut*

```
dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            calculated;
        value           uniform 0;
    }

    outlet
    {
        type            calculated;
        value           uniform 0;
    }


    top
    {
        type            nutkWallFunction;
        value           uniform 0;
    }


    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

Turb. therm. diff. *alphat*

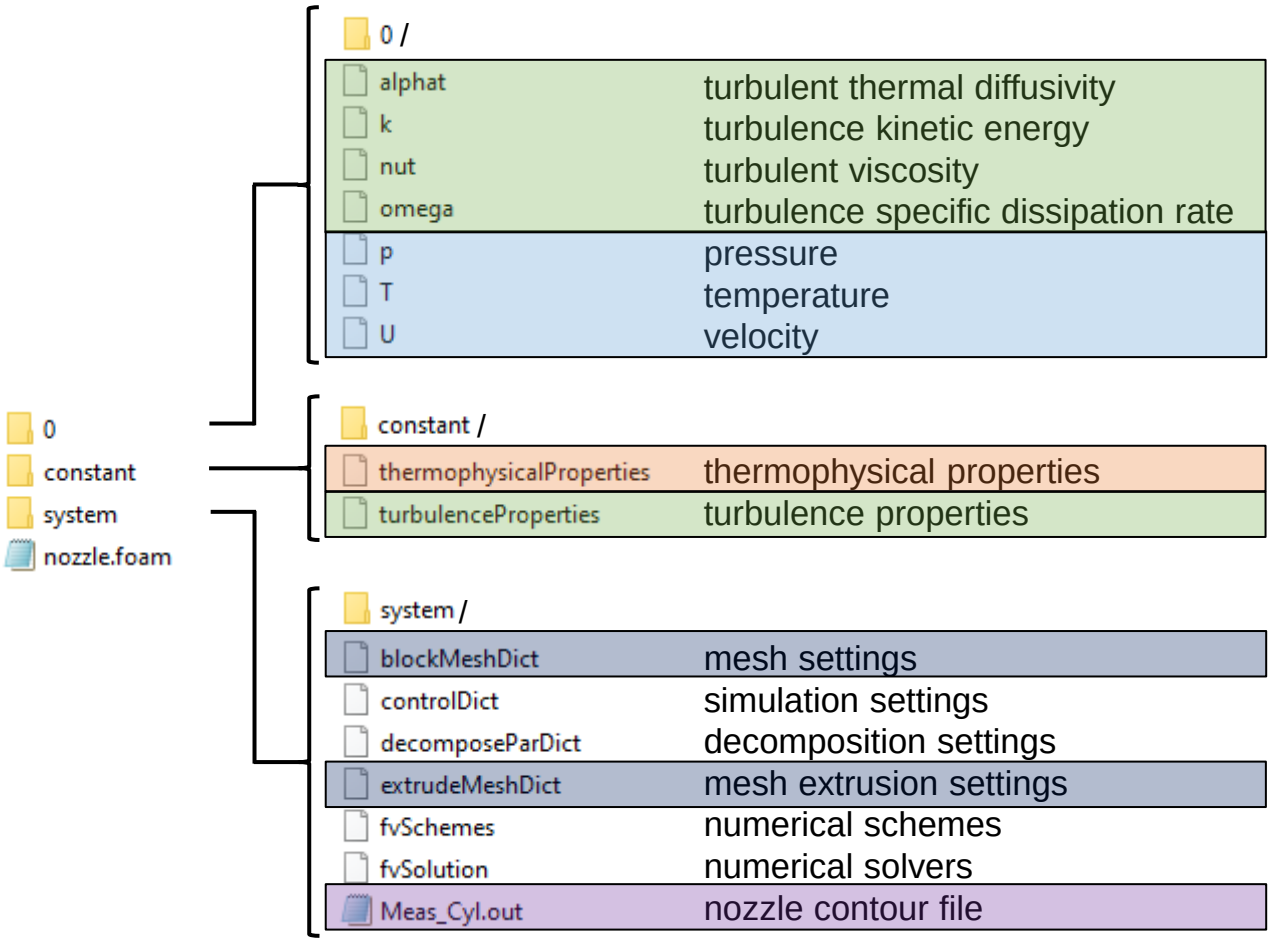
```
dimensions      [1 -1 -1 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            calculated;
        value           uniform 0;
    }

    outlet
    {
        type            calculated;
        value           uniform 0;
    }

    top
    {
        type            compressible::alphatWallFunction;
        value           $internalField;
    }

    axis
    {
        type            empty;
    }

    front
    {
        type            wedge;
    }

    back
    {
        type            wedge;
    }

}
```

Agenda (Part 1)

Pre-processing

1. Geometry creation
2. Mesh creation
3. Numerical setup
   a) Boundary conditions
   b) Turbulence model
   c) Real gas model
   d) Numerical settings

OpenFOAM folder structure



3.c) Real gas model

- The real gas model for hydrogen is selected in the ***thermophysicalProperties*** file

Real gas settings

```
thermoType
{
    type            hePsiThermo;
    mixture         pureMixture;
    transport       H2Transport;
    thermo          H2Thermo;
    equationOfState LeachmanH2Gas;
    specie          specie;
    energy          sensibleInternalEnergy;
}

mixture
{
    specie
    {
        molWeight       2.016;
    }

    equationOfState
    {
        Tc              33.145;
        Vc              0.064483;
        Pc              1296400.0;
    }

}
```

Developed real gas model

➢ GitLab link

Molar mass [g/mol]

Critical temperature [K]

Critical volume [m^3/kmol]

Critical pressure [Pa]

Ideal gas settings

```
thermoType
{
    type            hePsiThermo;
    mixture         pureMixture;
    transport       const;
    thermo          hConst;
    equationOfState perfectGas;
    specie          specie;
    energy          sensibleInternalEnergy;
}

mixture
{
    specie
    {
        molWeight       2.016;
    }
    thermodynamics
    {
        Cp              14300;
        Hf              0;
    }
    transport
    {
        mu              0.89e-05;
        Pr              0.7;
    }
}
```

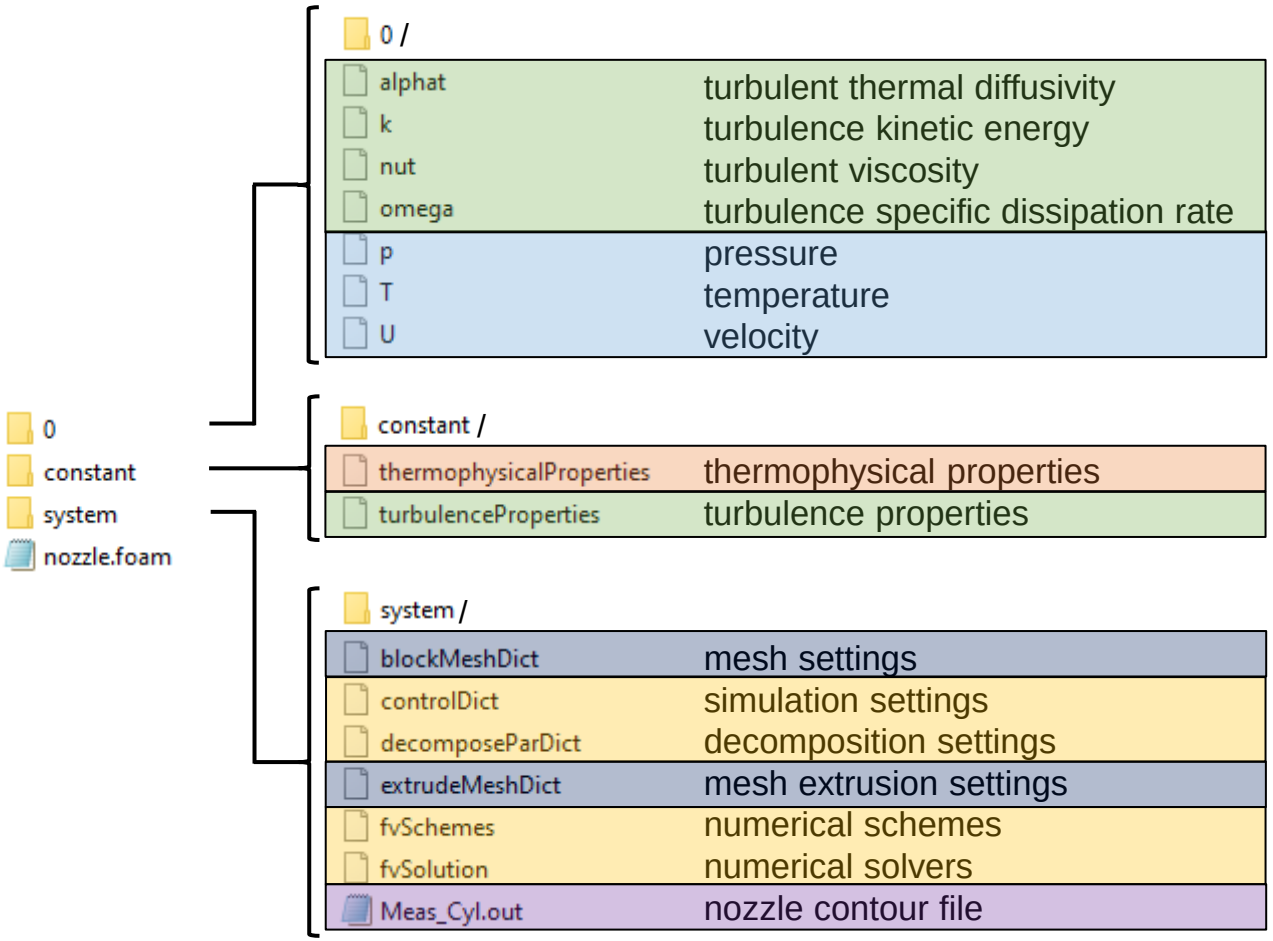
- can be used for low pressure cases

Agenda (Part 1)

Pre-processing

1. Geometry creation
2. Mesh creation
3. Numerical setup
   a) Boundary conditions
   b) Turbulence model
   c) Real gas model
   d) Numerical settings

OpenFOAM folder structure



3.d) Numerical settings

- In the ***controlDict*** file, the main case controls are set (e. g. time and write settings, functions objects, etc.)

```
application     sonicFoam;

startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         10e-04;

deltaT          0.25e-8;

writeControl    adjustableRunTime;
writeInterval   5e-05;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
adjustTimeStep          no;
maxCo                   0.8;
maxDeltaT               1e-03;
```

Compressible flow solver

Start and end time

Time step of the simulation

- How can we estimate an appropriate magnitude for the time step *Δt*?
  - Start with CFL condition in the axial direction at the nozzle throat:

$$CFL = u^* \cdot \frac{\Delta t}{\Delta z} \leq 1$$

$$\rightarrow \Delta t \leq \frac{\Delta z}{u^*} \quad \approx 1.25 \cdot 10^{-8}\ \mathrm{s}$$

Velocity at nozzle throat

$$u^* = \sqrt{\frac{2\kappa}{\kappa+1} R_M T_0} \approx 1200\,\frac{\mathrm{m}}{\mathrm{s}}$$

Cell size in the axial direction

$$\Delta z = \frac{l_z}{cz} = \frac{9\ \mathrm{mm}}{600} = 1.5 \cdot 10^{-5}\ \mathrm{m}$$

- This value can be used as a starting point

➢ Chosen time step ($0.25 \cdot 10^{-8}$ s) is smaller due to stability reasons

3.d) Numerical settings

- In the ***controlDict*** file, the main case controls are set (e. g. time and write settings, functions objects, etc.)

```
application     sonicFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         10e-04;
deltaT          0.25e-8;
writeControl    adjustableRunTime;
writeInterval   5e-05;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
adjustTimeStep          no;
maxCo                   0.8;
maxDeltaT               1e-03;
```

- Compressible flow solver
- Start and end time
- Time step of the simulation
- Adjusts time step to coincide with *writeInterval* (for automatic time stepping)
- Controls number of stored time directories (set to *0* to disable this)
- Specifies data file format and precision, and whether they are compressed or not
- Format and precision of the naming of the time directories
- Allows user to modify parameters during simulation
- Control to adjust time step during simulation (here: not active)

3.d) Numerical settings

- Functions objects are specified under ***functions*** (included in ***controlDict*** file)

```
functions
{
        massFlowRate
        {
                type                    surfaceFieldValue;
                functionObjectLibs      ("libfieldFunctionObjects.so");
                log                     no;
                writeControl            runTime;
                writeInterval           $deltaT;
                writeFields             no;
                regionType              patch;
                name                    inlet;
                operation               sum;
                fields
                (
                        phi
                );
        }

        MachNumberRealH2
        {
                type                    MachNoRealH2;
                libs                    ("libfieldmyFunctionObjects.so");
                executeControl          timeStep;
                writeControl            writeTime;
        }

        rhofunc
        {
                type                    writeObjects;
                libs                    ("libutilityFunctionObjects.so");
                executeControl          timeStep;
                writeControl            writeTime;
                objects
                (
                        "rho"
                );
        }
}
```

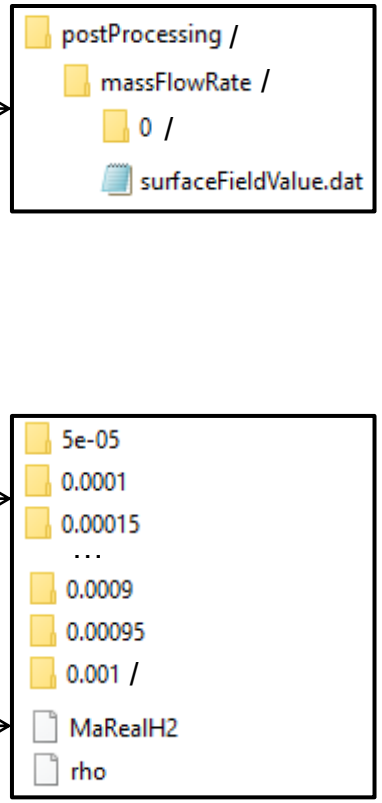
Calculates mass flow rate at the inlet patch in every time step (*deltaT*) and stores it in a text file

Note: *this is a user-defined function and library (suiting our developed real gas model)*

Calculates Mach number distribution at defined time intervals (*writeInterval*) and stores it in the time directories

Calculates density distribution at defined time intervals (*writeInterval*) and stores it in the time directories

Storage location



3.d) Numerical settings

- In the ***fvSchemes*** file, the numerical schemes are set

```
ddtSchemes
{
    default         Euler;
    ddt(phi)        Euler;
}
gradSchemes
{
    default         Gauss linear;
}
divSchemes
{
    default         none;
    div(phi,U)      Gauss limitedLinearV 1;
    div(phi,e)      Gauss limitedLinear 1;
    div(phi,p)      Gauss limitedLinear 1;
    div(phi,K)      Gauss limitedLinear 1;
    div(phi,p)      Gauss limitedLinear 1;
    div(phi,k)      Gauss upwind;
    div(phi,omega)  Gauss upwind;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}
laplacianSchemes
{
    default         Gauss linear corrected;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default         corrected;
}
wallDist
{
        method                  meshWave;
}
```

Time schemes $\frac{\partial}{\partial t}(\phi)$

Gradient schemes $\nabla(\phi)$

Divergence schemes $\nabla \cdot (Q)$

Laplacian schemes $\nabla^2(\phi)$

Interpolation schemes

Surface-normal gradient schemes

Wall distance calculation method

Keywords:

General:
- phi: Flux across cell faces
- Gauss: Gaussian integration

Time scheme:
- Euler: First order, implicit, bounded

Discretization schemes:
- linear: Second order, unbounded
- limitedLinear: First / second order, unbounded
- upwind: First order, bounded

Correction scheme:
- corrected: Second order, non-orthogonality

Remember:

- First order: bounded / stable but diffusive
- Second order: accurate but might oscillate

➢ Compromise between accuracy and stability

3.d) Numerical settings

- In the ***fvSolution*** file, solvers, tolerances, and algorithms are set

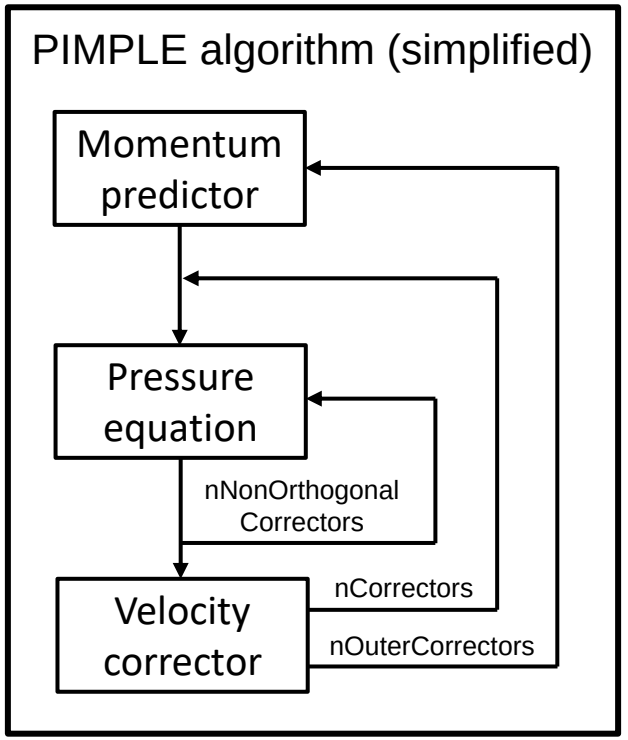
```
solvers
{
    "rho.*"
    {
        solver          diagonal;
    }

    "p.*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-08;
        relTol          0;
    }

    "(U|e|R).*"
    {
        $p;
        tolerance       1e-07;
    }

    "(k|omega).*"
    {
        $p;
        tolerance       1e-08;
    }
}

PIMPLE
{
    nOuterCorrectors 2;
    nCorrectors      1;
    nNonOrthogonalCorrectors 0;
}
```

Diagonal solver for explicit systems

Symmetric Gauss-Seidel solver

PIMPLE algorithm

- Pressure-velocity coupling algorithm fulfilling momentum and continuity equation in each time step



nOuterCorrectors:	number of momentum predictor (outer) loops
nCorrectors:	number of velocity corrector (inner) loops
nNonOrthogonalCorrectors:	number of pressure (inner) loops

3.d) Numerical settings

Run simulation

Option 1

Serial run:

1. Start simulation and write a log file
   sonicFoam > log &

Option 2

Parallel run:

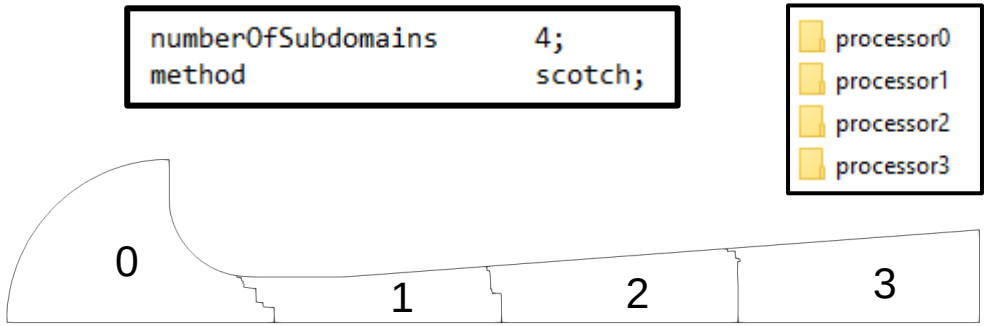
1. Decompose flow domain into subdomains
   decomposePar &
2. Start parallel simulation and write log file
   mpirun -np 4 sonicFoam -parallel > log &
3. Reconstruct flow domain (after simulation)
   reconstructPar &

Simulation running

- Let‘s have a coffee break until the simulation is completed ☺

decomposeParDict

```
numberOfSubdomains      4;
method                  scotch;
```



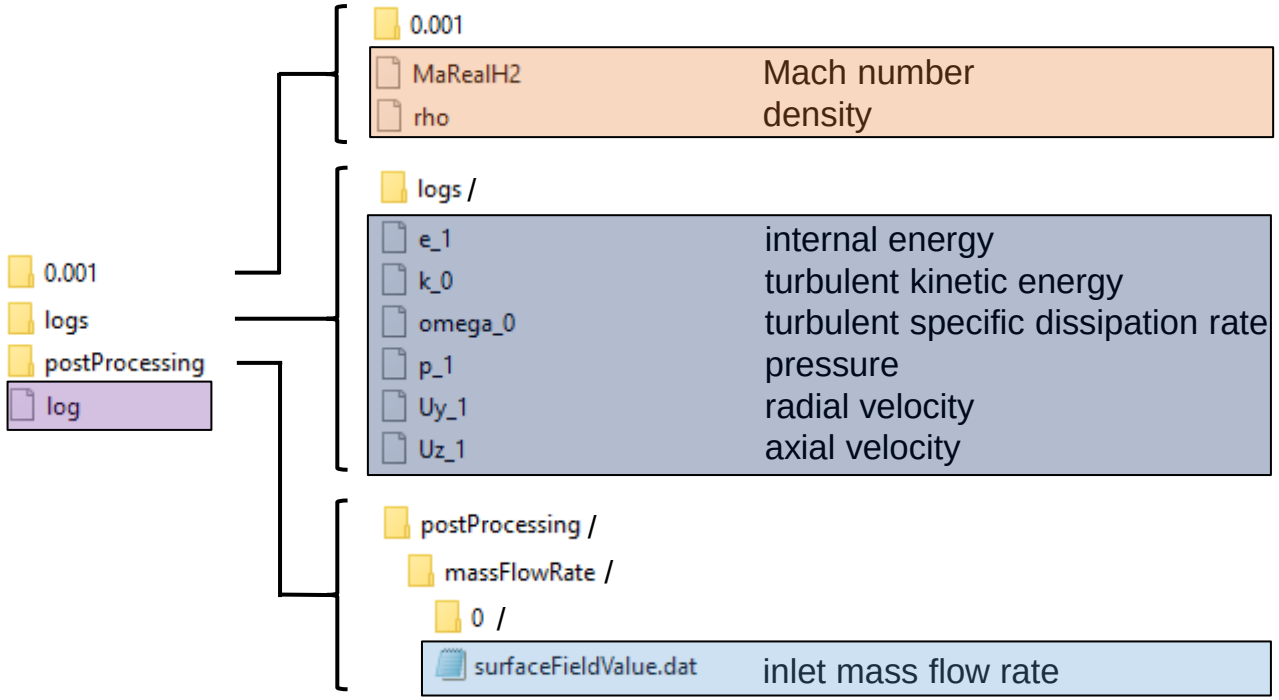
Five-minute break

Agenda (Part 2)

Post-processing

4.	Log file
5.	Convergence
a)	Residuals
b)	Mass flow rate
6.	Discharge coefficient
7.	Flow field
a)	Surface plots
b)	Line plots

OpenFOAM folder structure

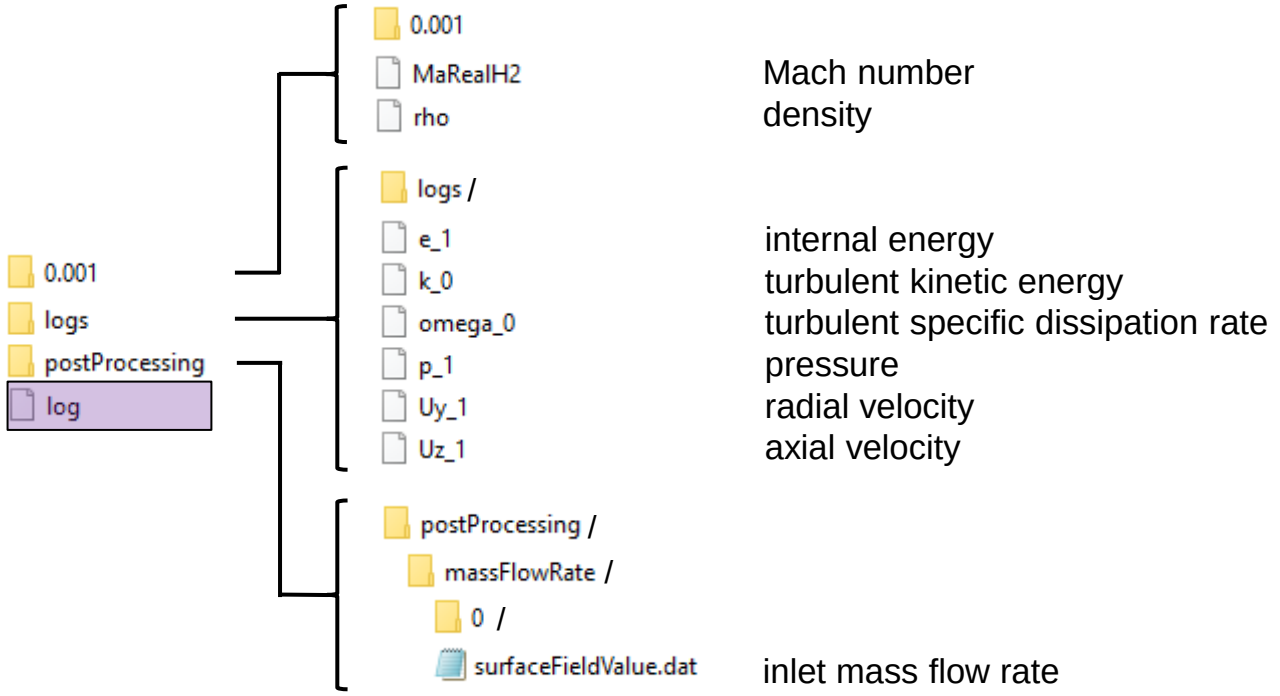


Agenda (Part 2)

Post-processing

4. Log file
5. Convergence
   a) Residuals
   b) Mass flow rate
6. Discharge coefficient
7. Flow field
   a) Surface plots
   b) Line plots

OpenFOAM folder structure



4. Log file

- Let‘s have a look at the log file:
  Command: *vim log*

Time step →

Courant number →

1st PIMPLE outer loop

2nd PIMPLE outer loop

Run time →

Writing function objects →

Run completed →

```
Time = 0.001

Courant Number mean: 0.0731337 max: 0.596613
diagonal:  Solving for rho, Initial residual = 0, Final residual = 0, No Iterations 0
PIMPLE: iteration 1
smoothSolver:  Solving for Ux, Initial residual = 0.00512428, Final residual = 5.13094e-08, No Iterations 3
smoothSolver:  Solving for Uy, Initial residual = 0.00045734, Final residual = 1.74349e-08, No Iterations 4
smoothSolver:  Solving for Uz, Initial residual = 4.1337e-05, Final residual = 2.2945e-08, No Iterations 3
smoothSolver:  Solving for e, Initial residual = 9.59416e-05, Final residual = 6.25853e-08, No Iterations 3
smoothSolver:  Solving for p, Initial residual = 5.04294e-05, Final residual = 4.26458e-09, No Iterations 5
diagonal:  Solving for rho, Initial residual = 0, Final residual = 0, No Iterations 0
time step continuity errors : sum local = 3.56692e-09, global = 3.8627e-10, cumulative = -9.03055e-05
PIMPLE: iteration 2
smoothSolver:  Solving for Ux, Initial residual = 2.36959e-05, Final residual = 1.1483e-08, No Iterations 2
smoothSolver:  Solving for Uy, Initial residual = 3.60669e-05, Final residual = 3.54568e-08, No Iterations 2
smoothSolver:  Solving for Uz, Initial residual = 2.14832e-06, Final residual = 4.22986e-08, No Iterations 1
smoothSolver:  Solving for e, Initial residual = 2.73181e-05, Final residual = 2.02109e-08, No Iterations 2
smoothSolver:  Solving for p, Initial residual = 2.30766e-05, Final residual = 5.29608e-09, No Iterations 3
diagonal:  Solving for rho, Initial residual = 0, Final residual = 0, No Iterations 0
time step continuity errors : sum local = 4.42968e-09, global = 1.64049e-10, cumulative = -9.03053e-05
smoothSolver:  Solving for omega, Initial residual = 2.85294e-05, Final residual = 1.13305e-09, No Iterations 3
smoothSolver:  Solving for k, Initial residual = 6.99569e-05, Final residual = 1.69573e-09, No Iterations 4
ExecutionTime = 60469.4 s  ClockTime = 61141 s

    functionObjects::MachNoRealH2 MachNumberRealH2 writing field: MaRealH2
writeObjects rhofunc write:
    writing object rho
End

Finalising parallel run
```

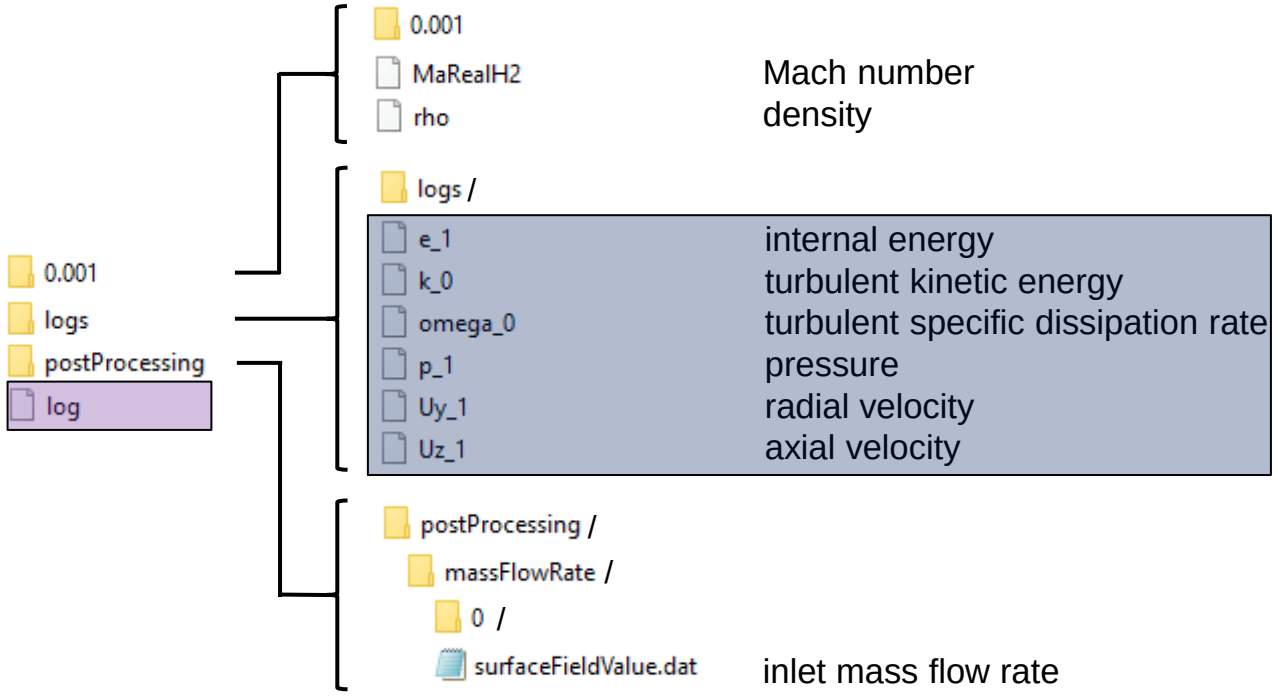
$\approx 17\ \mathrm{h}$

Agenda (Part 2)

Post-processing

4. Log file
5. Convergence
   a) Residuals
   b) Mass flow rate
6. Discharge coefficient
7. Flow field
   a) Surface plots
   b) Line plots

OpenFOAM folder structure

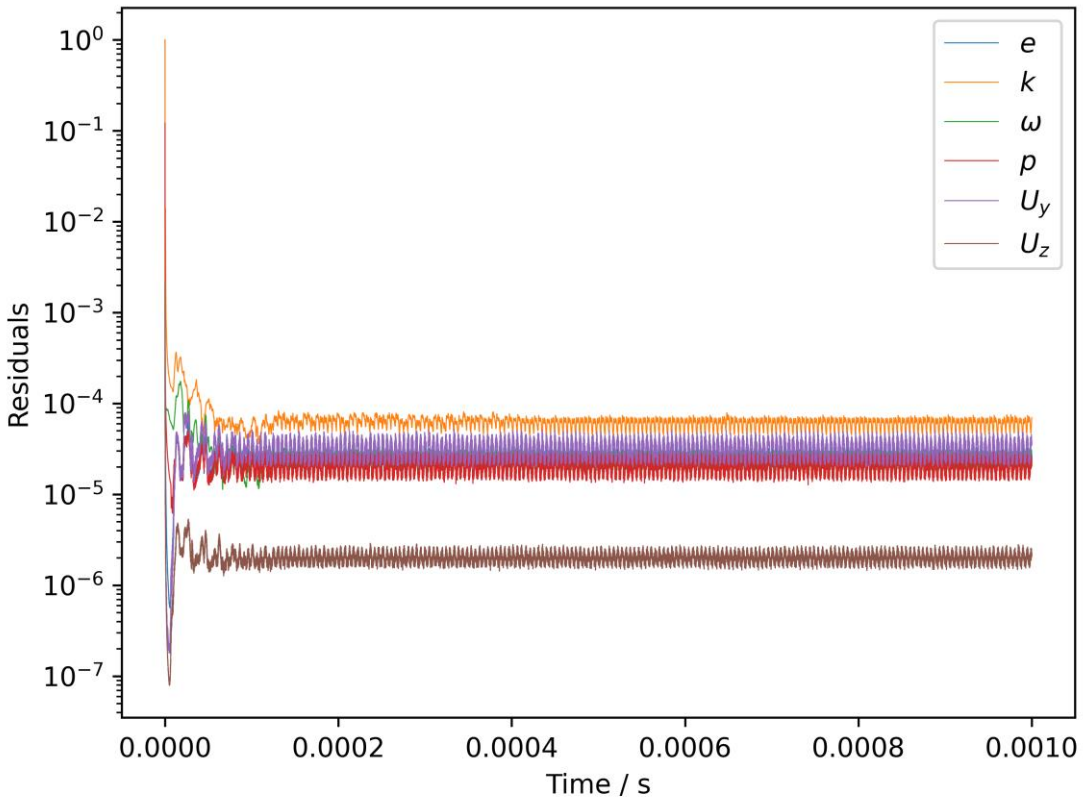


5.a) Residuals

- Let‘s check the convergence of the simulation
- First, we extract the residual data from the log file:
  Command: *foamLog log &*

- This generates a */logs* folder
- ➢ Let’s plot the residuals from the output data

- ➢ Residuals quicky drop and slightly oscillate around a constant value

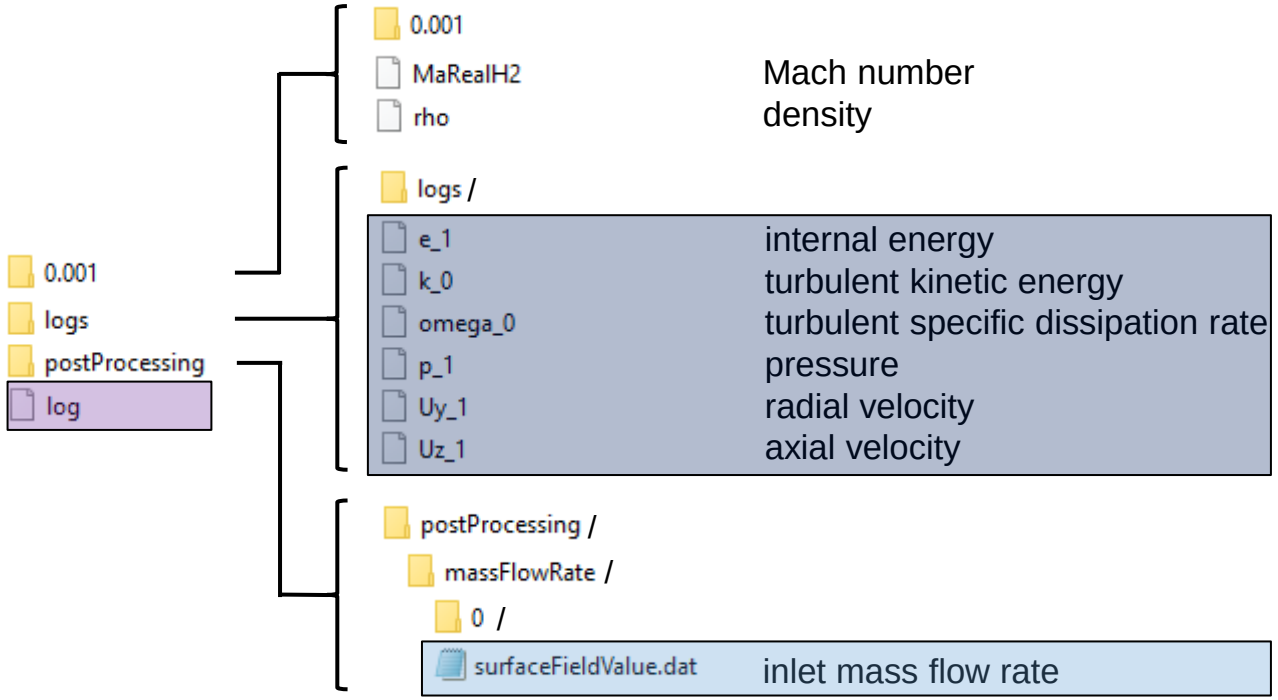


Agenda (Part 2)

Post-processing

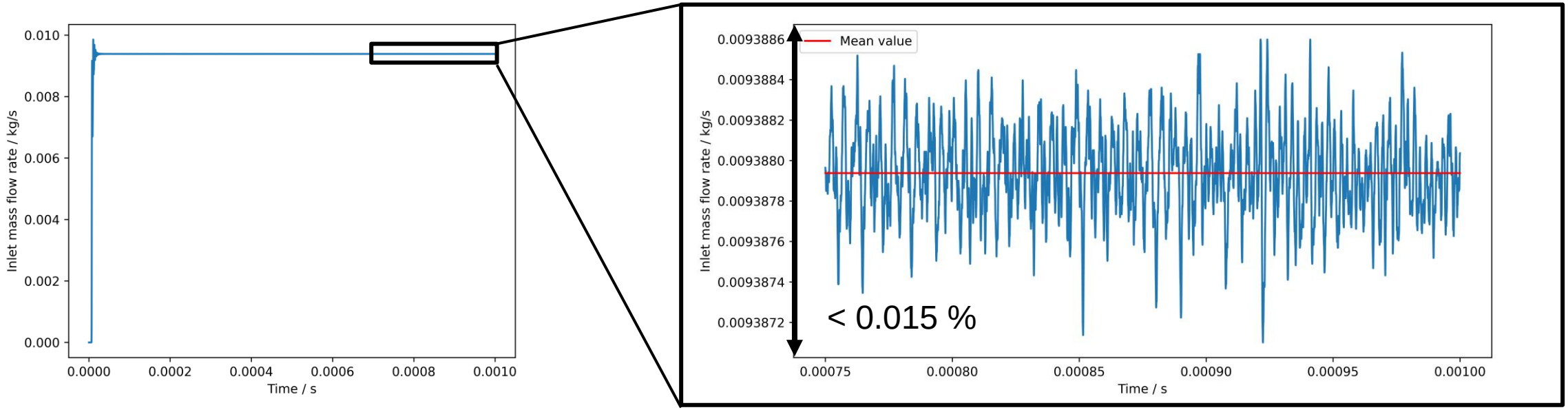
4. Log file
5. Convergence
   a) Residuals
   b) Mass flow rate
6. Discharge coefficient
7. Flow field
   a) Surface plots
   b) Line plots

OpenFOAM folder structure



5.b) Mass flow rate

- Let‘s see how the mass flow rate is establishing
- ➢ We monitored the mass flow rate at the inlet



Mean value

$$\dot{m}_{CFD} \approx 0.00938793 \frac{\mathrm{kg}}{\mathrm{s}}$$

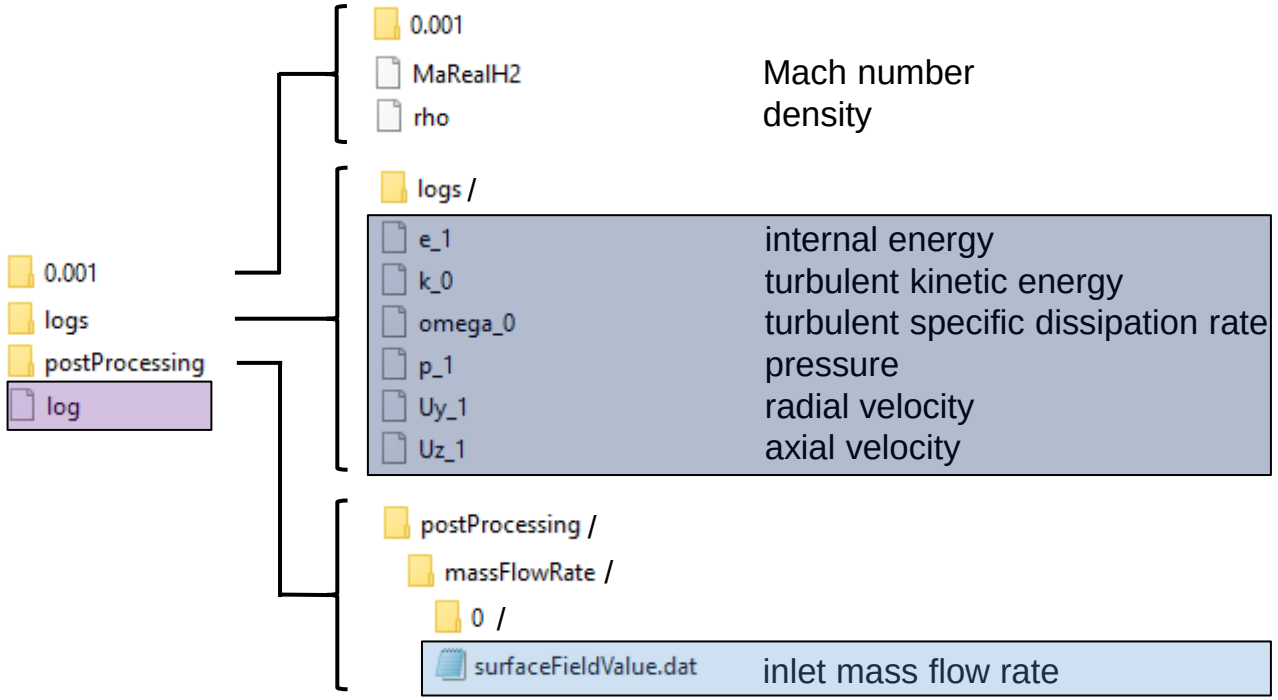
- ➢ Mass flow rate only varies within 0.015 % margin

Agenda (Part 2)

Post-processing

4. Log file
5. Convergence
   a) Residuals
   b) Mass flow rate
6. Discharge coefficient
7. Flow field
   a) Surface plots
   b) Line plots

OpenFOAM folder structure



6. Discharge coefficient

- Let‘s calculate the discharge coefficient

$$C_D = \frac{\dot{m}_{CFD}}{\dot{m}_{th}}$$

$$\dot{m}_{th} = C^* A^* \frac{p_0}{\sqrt{R_M T_0}}$$

Area of CFD domain

Ideal gas

$$C_i^* = \sqrt{\kappa \cdot \left(\frac{2}{\kappa+1}\right)^{\frac{\kappa+1}{\kappa-1}}}$$

$$\boxed{\rightarrow C_{D,i} \approx 0.9721}$$

Real gas

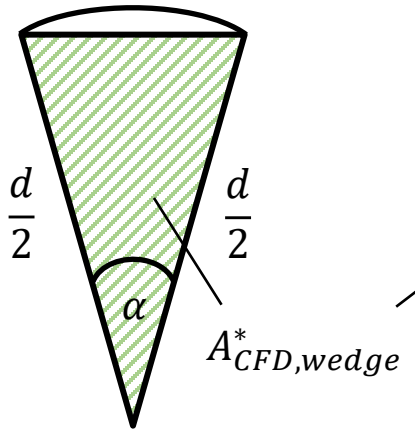
(From REFPROP)

$$C_r^* = 0.67155$$

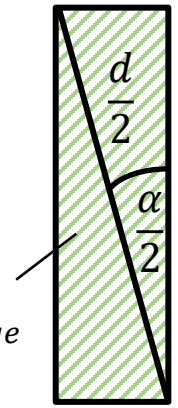
$$\boxed{\rightarrow C_{D,r} \approx 0.9912}$$

Relation

$$C_{D,r} = \frac{C_i^*}{C_r^*} C_{D,i}$$



$$b = \frac{d}{2} \sin\left(\frac{\alpha}{2}\right)$$



$$h = \frac{d}{2} \cos\left(\frac{\alpha}{2}\right)$$

$$A^*_{CFD,wedge} = b \cdot h = \sin\left(\frac{\alpha}{2}\right) \cos\left(\frac{\alpha}{2}\right) \frac{d^2}{4}$$

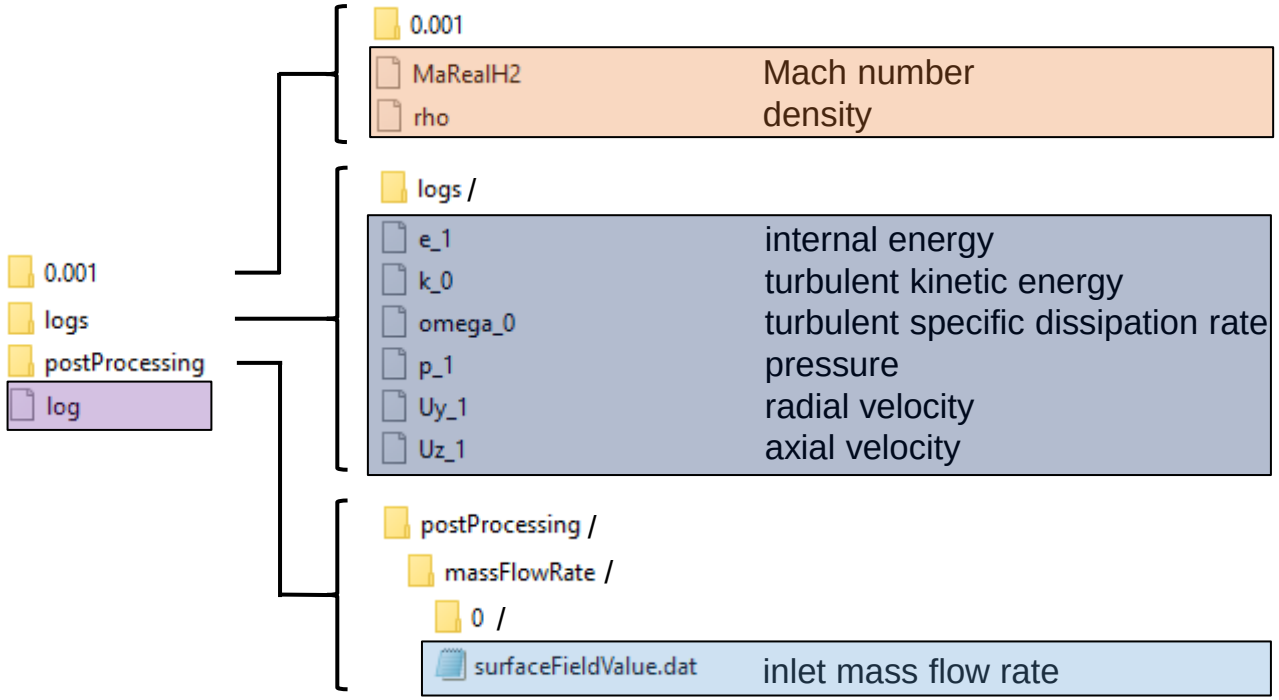
$$\rightarrow A^*_{CFD} = \frac{360}{\alpha} A^*_{CFD,wedge}$$

Agenda (Part 2)

Post-processing

4.	Log file
5.	Convergence
	a) Residuals
	b) Mass flow rate
6.	Discharge coefficient
7.	Flow field
	a) Surface plots
	b) Line plots

OpenFOAM folder structure



7.a) Surface plots

Mach number function for real gas hydrogen

- We introduced a specific function for the Mach number for our real gas model for hydrogen

```
MachNumberRealH2
{
        type                    MachNoRealH2;
        libs                    ("libfieldmyFunctionObjects.so");
        executeControl          timeStep;
        writeControl            writeTime;
}
```

➢ Why is this necessary?

- In OpenFOAM (OF), the Mach number function is defined as follows:

$$M_{OF} = \frac{|u|}{\sqrt{\kappa_{ideal}\frac{p}{\rho}}} \qquad \kappa_{ideal} = \frac{c_p}{c_v}$$

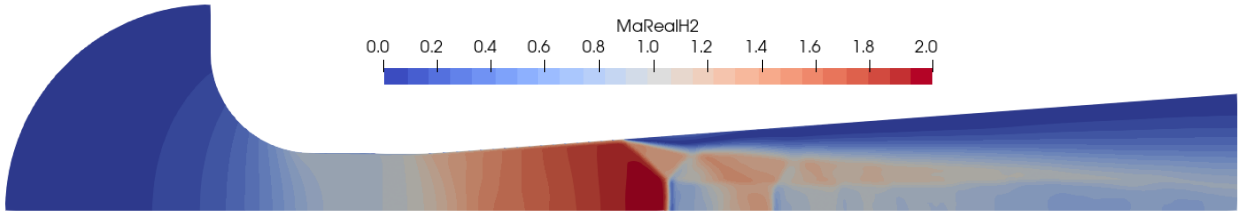
- Here, the ideal isentropic exponent is used

- For a real gas, the isentropic exponent is defined as follows:

$$\kappa_{real} = \frac{c_p}{c_v}\left(\frac{\partial p}{\partial \rho}\right)_T \frac{\rho}{p}$$

- Our Mach number function directly uses the speed of sound correlation a_{real}:

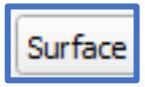
$$M_{real} = \frac{|u|}{a_{real}}$$



➢ Let‘s see how to create surface plots like this

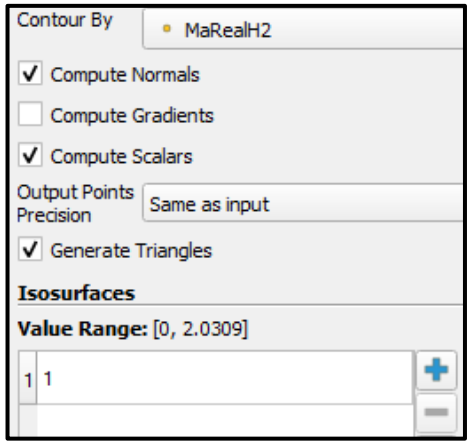
7.a) Surface plots

ParaView to visualize the flow field



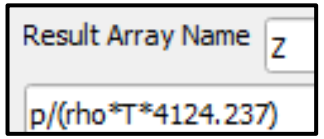
1. Surface plots

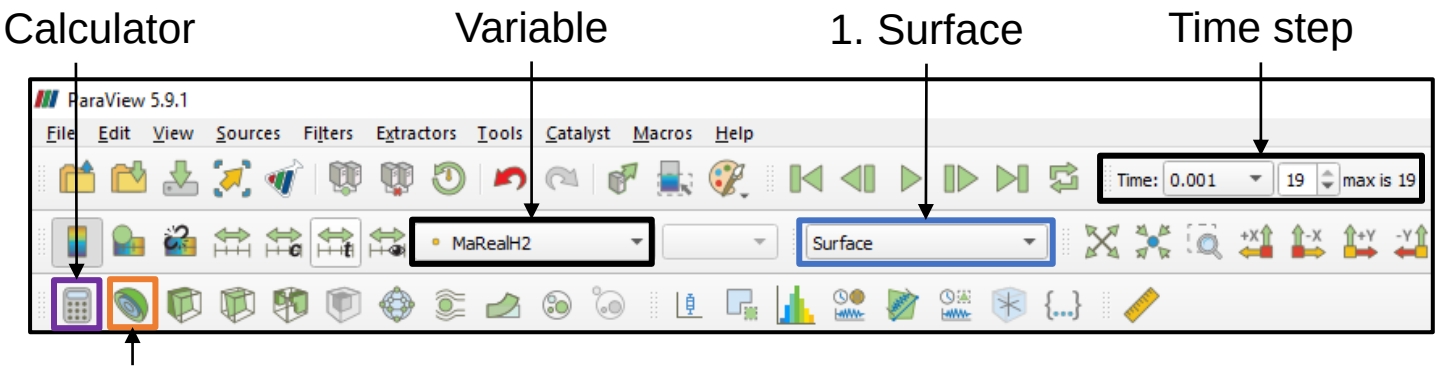
2. Contour plots

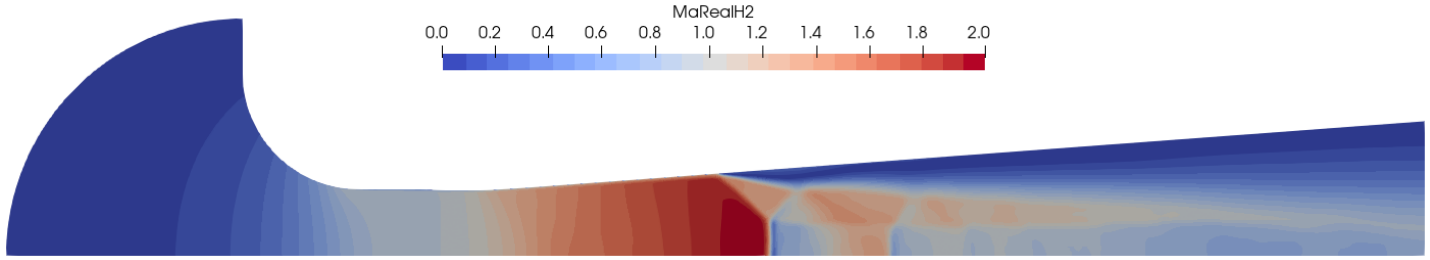


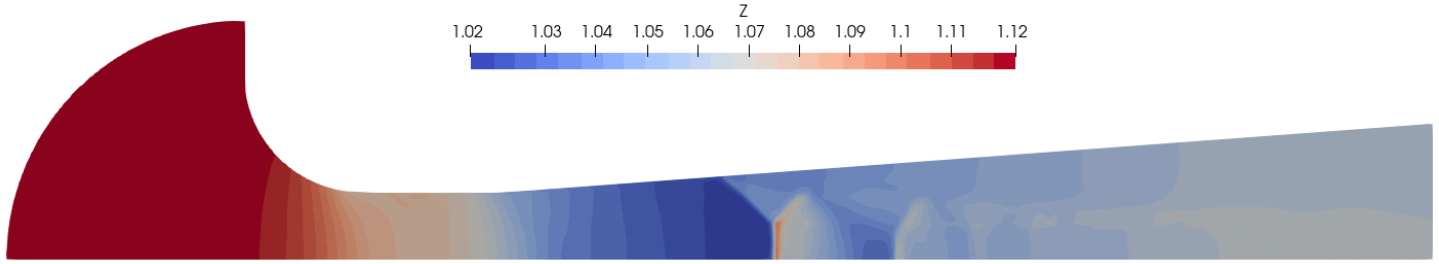
3. Calculated variables

Compressibility factor *Z*









7.a) Surface plots

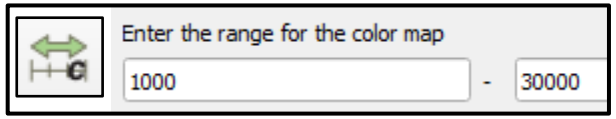
Numerical Schlieren

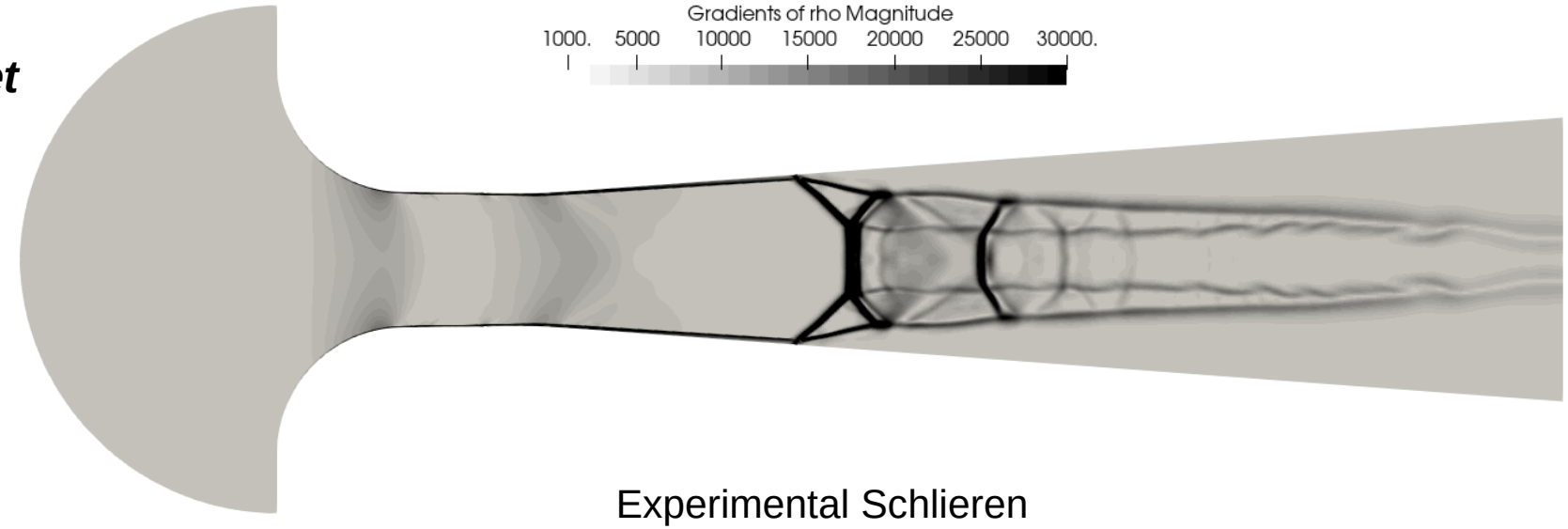
Gradient of Unstructured DataSet filter

1. Set density gradient

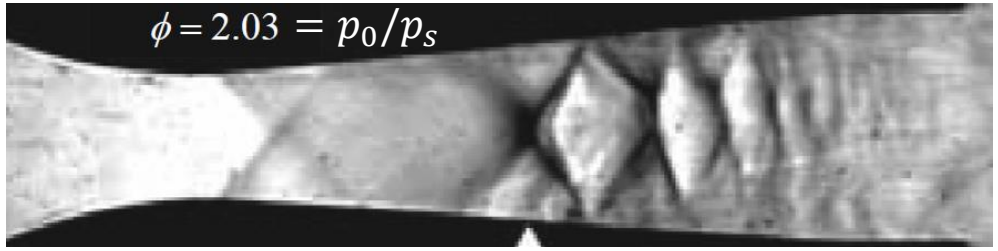
2. Apply X Ray colormap

3. Adjust Min. and Max. values





Experimental Schlieren
(for qualitative comparison)



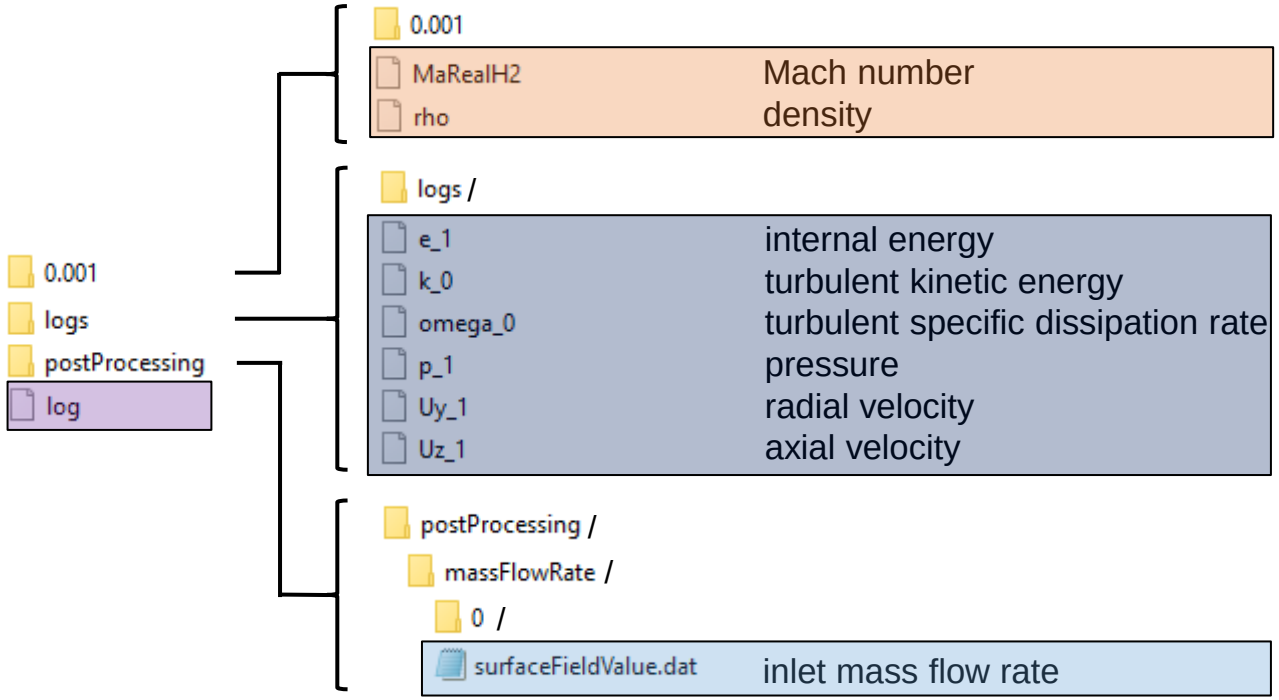
Source: S. Matsuo et al., "*Effects of Supersonic Nozzle Geometry on Characteristics of Shock Wave Structure*", Open Journal of Fluid Dynamics, Vol. 2 No. 4A, 2012.

Agenda (Part 2)

Post-processing

4.	Log file
5.	Convergence
a)	Residuals
b)	Mass flow rate
6.	Discharge coefficient
7.	Flow field
a)	Surface plots
b)	Line plots

OpenFOAM folder structure

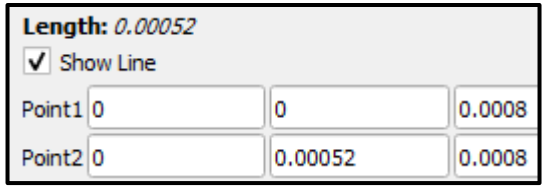


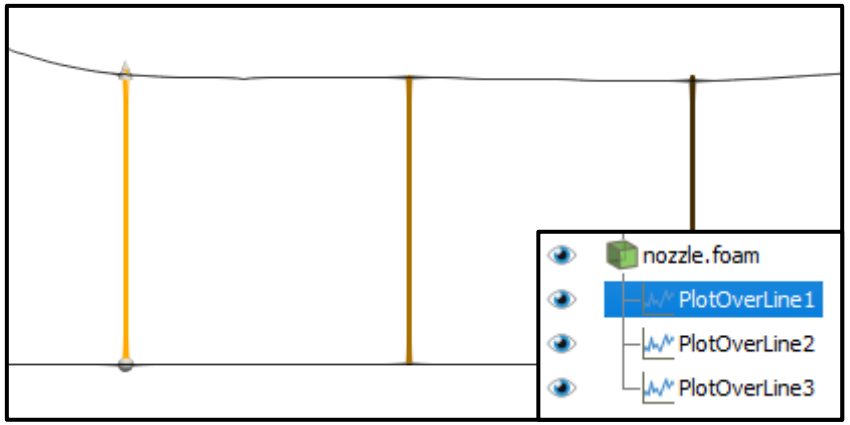
7.b) Line plots

Plot velocity over nozzle throat height

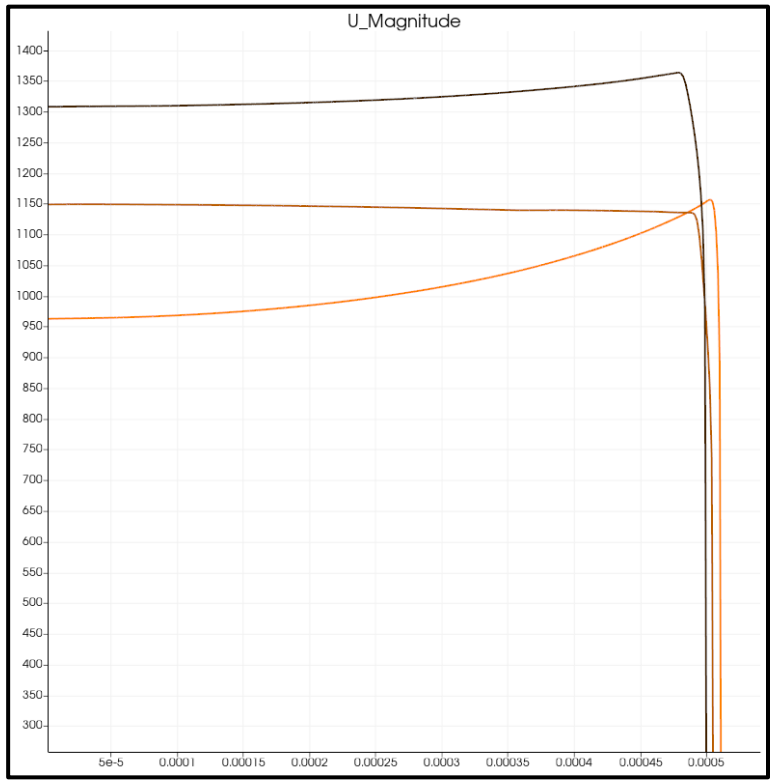
Plot Over Line filter

1. Define the line location

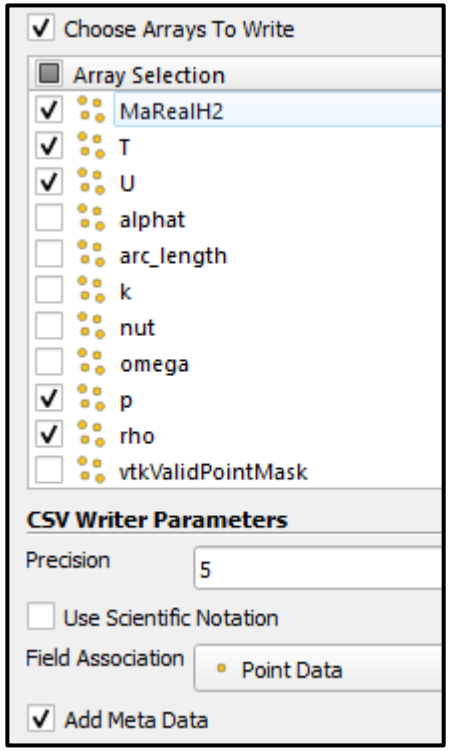




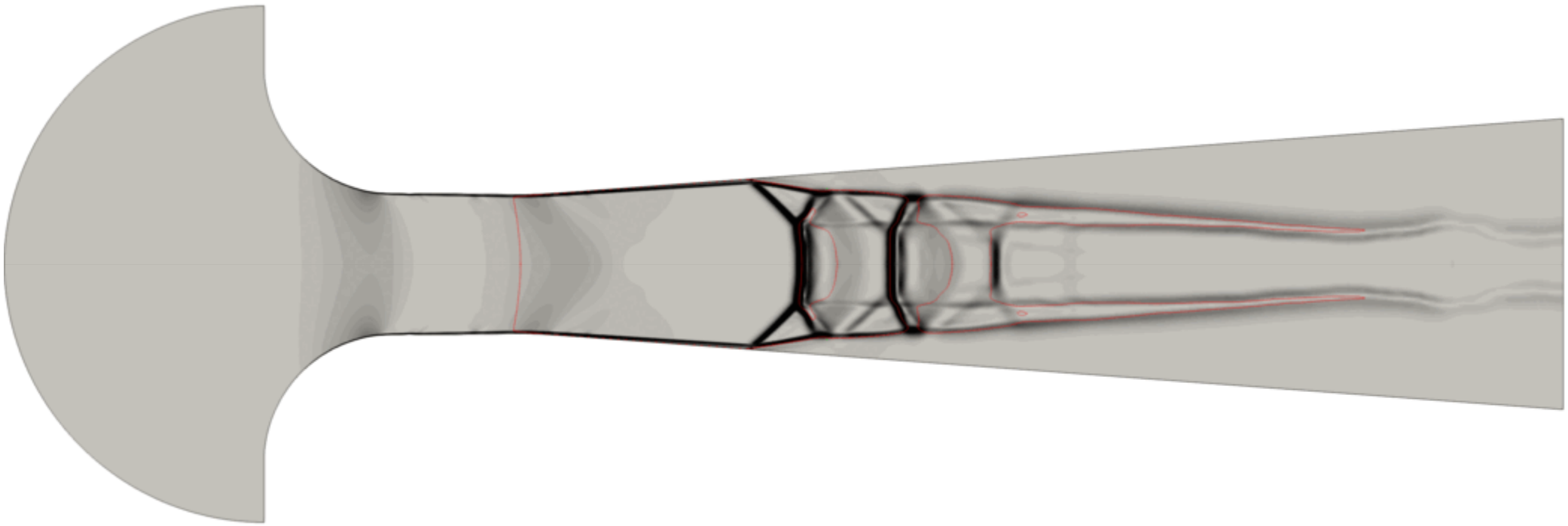
2. Select the velocity



3. Save line data as text file



Thank you!



MetHyInfra

Internal

PTB
Physikalisch-Technische Bundesanstalt
Braunschweig und Berlin

CESAME-exadébit SA

Justervesenet

METAS

RISE
Research Institutes of Sweden

TÜV SÜD
National Engineering Laboratory

VSL
Dutch Metrology Institute

external

THERMODYNAMIK
RUHR UNIVERSITÄT BOCHUM
RUB

University of Ljubljana
Faculty of Mechanical Engineering

esi
get it right®
MECAS ESI s.r.o.

Imperial College London

MAXIMATOR®
Maximum Pressure.

DNV

Perfection in fluids.
The right flow by German engineering.
ep-e.com

Ingenieurbüro
T. Steuer
its-tech.de

Bronkhorst®
Flow Technology

This project (20IND11 MetHyInfra) has received funding from the EMPIR programme co-financed by the Participating States and from the European Union's Horizon 2020 research and innovation programme.

EMPIR EURAMET

The EMPIR initiative is co-funded by the European Union's Horizon 2020 research and innovation programme and the EMPIR Participating States

MetHyInfra

Sebastian Weiss

Working Group 8.41 “Modelling and Simulation” and
Working Group 1.45 “Hydrogen Quantity Metering”
Physikalisch-Technische Bundesanstalt (PTB)
Berlin, Germany

sebastian.weiss@ptb.de